TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JUHA HANKKILA

LIIKKUVAN MYYJÄN TABLET-SOVELLUKSEN TOTEUTTAMINEN WINDOWS 8 -ALUSTALLE

Diplomityö

Tarkastaja: professori Tommi Mikkonen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekuntaneuvoston
kokouksessa 12. elokuuta 2015

TIIVISTELMÄ

HANKKILA, JUHA: Liikkuvan myyjän tablet-sovelluksen toteuttaminen Windows 8 -alustalle
Tampereen teknillinen yliopisto
Diplomityö, 47 sivua, 2 liitesivua
Joulukuu 2015
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Tommi Mikkonen
Avainsanat: tablet, Windows 8, .NET, C#, Windows Runtime, WinRT

Teknologian kehittymisen ja uusien laitetyyppien markkinoille tulon myötä yrityksissä on pohdittu eri tapoja niiden hyödyntämiseen oman liiketoiminnan kehittämisessä. Tietotekniikan tapauksessa tämä näkyi PC-tietokoneiden yleistymisenä yrityskäytössä viime vuosituhannen lopulla. Kannettavat tietokoneet mahdollistivat samojen tehtävien hoitamisen toimiston ulkopuolella. Myöhemmin älypuhelimet ja PDA-laitteet ovat tarjonneet aivan uudenlaisen tavan toimia mobiilisti. Eri tehtäviä varten on pystytty luomaan mobiilisovelluksia, joilla tarvittavat toimenpiteet saadaan hoidettua helposti, nopeasti ja juuri siellä, missä on tarve. Uusimpana alustatyyppinä tabletit tehostavat mobiilia käyttökokemusta entisestään mahdollistaen monipuolisempien sovellusten kehittämisen.

Liikkuvat myyjät kohtaavat työssään päivittäin monia haasteita ja tarpeita, joita ovat muun muassa asiakaskäyntien suunnittelu, tuotteiden esittely ja markkinointi, asiakastietojen ylläpito, tilausten tekeminen sekä samalla ylimyynnin välttäminen. Näihin on mahdollista tarjota tietoteknisiä ratkaisuja, mikä oli yksi käsitellyn sovellusprojektin keskeisimmistä tavoitteista ja samalla sen suurimmista haasteista. Jotta myyjä pystyy toimimaan tehokkaasti ja lisäämään myyntiä, muiden usein käytettyjen toimintojen tavoin myös tilausten teon tulee onnistua nopeasti. Koska sovelluksen käsittelemät tietomäärät ovat todella suuria, onnistunut toteutus edellyttää asian huomioimista jo sovelluksen arkkitehtuuria suunniteltaessa.

Tässä diplomityössä käsitellään Eatech Portable Salesin, liikkuvan myyjän tablet-sovelluksen toteuttaminen Windows 8 -alustalle. Ohjelmistoprojekti alkoi vuoden 2012 lopulla ja sitä jatkokehitetään edelleen. Kehityksen myötä myös sovelluksen arkkitehtuuriin on tullut muutoksia. Työssä kuvataan käytetyt arkkitehtuuriratkaisut perustellen näiden valintaan johtaneet syyt. Tämän työn avulla sovelluksen kehitykseen tulevaisuudessa osallistuvat henkilöt oppivat nopeammin sovelluksen arkkitehtuurin, siinä käytetyt menetelmät ja tietävät valintojen taustalla olleet tekijät. Lisäksi esitellään alustan keskeisimmät piirteet ja sen sovelluskehitysmenetelmät, joten työstä voi olla hyötyä myös vastaavia Windows 8 tablet -sovelluksia kehittäville.

Asiakkailta ja sovelluksen käyttäjiltä saadun palautteen perusteella työssä kuvattu ohjelmistoprojekti voidaan todeta onnistuneeksi. Asiakkaat ovat kertoneet pystyneensä tehostamaan liiketoimintaansa ratkaisun avulla. Sovellus täytti kaikki sille asetetut vaatimukset, ja se valittiin lisäksi Microsoftin järjestämän *App Awards* -sovelluskilpailussa parhaaksi yrityssovellukseksi.

ABSTRACT

HANKKILA, JUHA: Designing a Tablet Application for Travelling Salesman Using Windows 8
Tampere University of Technology
Master of Science Thesis, 47 pages, 2 Appendix pages
December 2015
Master's Degree Programme in Information Technology
Major: Software Engineering
Examiner: Professor Tommi Mikkonen
Keywords: tablet, Windows 8, .NET, C#, Windows Runtime, WinRT

As technology has advanced and new platforms have emerged, businesses have started considering using them to increase their profits. In the field of Information Technology, this showed first as PC-computers becoming more common in businesses at the turn of the century. Laptop computers made it possible to perform the same tasks outside of office, after which smart phones and PDA-devices offered brand new ways to work mobile. As a result, it became possible to create mobile applications for different purposes so that the necessary tasks could be performed easily, swiftly and exactly where and when needed. The newest platform – tablets – made even more efficient applications possible.

Travelling salesmen face multiple challenges and needs in their daily work ranging from planning and preparation of customer visits, marketing and presenting products and keeping the customer records up-to-date to processing orders. Developing IT solutions for these different functions was one of the main aims as well as the biggest challenges in the discussed software project. In order to be able to be work efficiently and increase sales, the salesmen have to be able to process orders quickly. As the amount of data processed by the application is huge, a successful implementation requires that it has already been considered when planning the software architecture.

The goal of this Master's Thesis is to discuss Eatech Portable Sales, a tablet application for travelling salesmen. The software project started towards the end of year 2012 and the its development has continued to this day. Further development has also meant changes to the software architecture. In this thesis, the architectural choices will be described and the reasons behind them discussed. This thesis will help future software developers to familiarize themselves with the software architecture, methods used as well as the factors lying behind the choices. In addition, the central characteristics of the platform and the methods of software development will be presented hoping that it will help the development of similar Windows 8 tablet applications in the future.

Based on the feedback from the clients and users of the application, the software project discussed in this thesis can be seen as successful. The clients have reported being able to increase their profits by using the application. All expectations set for the application were met and the application was chosen as the best business application in Microsoft's *App Awards* application competition.

ALKUSANAT

Haluan kiittää työnantajaani Eatech Oy:ta mielenkiintoisesta projektista ja hyvästä aiheesta diplomityölle. Projektin aikana opin paljon uutta tablet-kehityksestä ja käyttöliittymien suunnittelusta. Iso kiitos kuuluu myös professori Tommi Mikkoselle diplomityön ohjaamisesta. Kirjoitusprosessi eteni hänen ohjauksessaan johdonmukaisesti, eikä missään vaiheessa tullut suurempia ongelmia vastaan.

Lopuksi haluan kiittää tyttöystävääni Tiinaa, perhettäni ja ystäviäni, jotka ovat tukeneet, auttaneet ja kannustaneet minua opintojeni ja diplomityöprosessin aikana.

Tampereella 6.11.2015

Juha Hankkila
juha.hankkila@eatech.fi, juha.hankkila@gmail.com

SISÄLLYSLUETTELO

1. JOHDANTO

Liikkuva myyjä joutuu kentällä toimiessaan suunnittelemaan myyntikäyntejä, joiden tueksi hän tarvitsee yrityksen asiakastietoja: missä asiakkaan toimitilat sijaitsevat, kehen olla yhteydessä, milloin asiakkaille on viimeksi tehty myyntiä, miten paljon ja mitä tuotteita on aikaisemmin myyty. Myyntikäynnillä tulee pystyä esittelemään tuotekatalogia, tekemään tilauksia ja näkemään mitä tuotteita löytyy heti varastosta tai milloin niitä olisi tulossa lisää. Tässä diplomityössä toteutetaan liikkuvan myyjän tablet-sovellus, joka sisältää kaiken myyjän kentällä tarvitsemat tiedot ja toiminnallisuudet samassa sovelluksessa.

Tablet-tietokoneet mahdollistavat aivan uudenlaisia käyttötapoja, joihin kannettavat tietokoneet ja pienet PDA-laitteet tai puhelimet eivät ole aikaisemmin soveltuneet. Kuluttajat ovat huomanneet tämän, mikä on näkynyt erityisesti tablettien yleistymisenä internet selailussa sekä viihdekäytössä. Tämän vuoksi tablet-valmistajat ovat keskittyneet erityisesti kuluttajamarkkinoihin. Myös yrityksissä monet ovat havainneet tablettien potentiaalin yrityksen liiketoiminnan tehostamisessa. Kuten kaikkien uusien keksintöjen kanssa, menee oma aikansa, että myös niitä osataan hyödyntää parhaalla mahdollisella tavalla.

Tämän työn yhteydessä toteutettu tablet-projekti sai alkunsa asiakkaan vanhan PDA-sovelluksen päivitystarpeista. Asiakkaalle oli aikaisemmin kehitetty hieman vastaava myyntijärjestelmä PDA-laitteilla käytettäväksi. Kyseiset laitteet olivat kuitenkin hitaita, eivätkä pienen ruutunsa takia soveltuneet kunnolla käyttötarkoitukseensa, joten asiakas toivoi sovelluksesta tablet-versiota. Isommalla ruudulla ja monipuolisemmalla alustalla sovellukseen saataisiin tuotua enemmän aineistoa ja sitä voisi käyttää myös tuote-esittelyihin muiden tehtävien ohella.

Koska Windows-käyttöjärjestelmät ovat yrityksissä käytetyimpiä ja samalla tutuimpia, päätettiin sovellus toteuttaa vasta julkaistulle Windows 8 -alustalle [1]. Windows 8 -käyttöjärjestelmälle sovelluksen olisi voinut toteuttaa perinteisenä työpöytäsovelluksena tai uutena Windows 8 -sovelluksena. Päädyimme jälkimmäiseen, koska sovelluksessa käytettävyys ja näyttävyys olivat keskeisessä asemassa ja Windows 8 -sovellukset tarjosivat parhaat edellytykset niihin.

Koska sovelluksen immateriaalioikeudet jäävät myös toteuttajalle, tämän kaltaiselle sovellukselle tuntui olevan kysyntää eikä markkinoilta vastaavaa ratkaisua löytynyt, päätettiin sovellus ottaa omaksi tuotteeksi. Ajan myötä *Eatech Portablesta* on kehitetty tuoteperhe. Liikkuvan myyjän työkalu tunnetaan nimellä *Eatech Portable Sales*, ja sitä on laajennettu julkaisun jälkeen muun muassa asiakkuudenhallintaominaisuuksilla. *Eatech Portable Services* on samasta tuotteesta huoltomiesten tarpeisiin räätälöity versio, jolla

kentällä liikkuvat huoltomiehet saavat tarvitsemansa tiedot työmääräimistä, pystyvät muun muassa kirjaamaan huomioita huoltotehtävästä ja kuittaamaan sen tehdyksi.

Projekti alkoi vuoden 2012 lopulla asiakkaan kanssa, omaa tuotetta ryhdyttiin rakentamaan 2013 kesällä ja sovellusta jatkokehitetään edelleen. Diplomityön kohteeksi sovellus valittiin vuoden 2013 lopulla, jolloin mietimme Joni Pakarisen kanssa toteuttamamme projektin aihepiirin jakamista kahdeksi opinnäytetyöksi. Aluksi harkitsimme myös yhteistä diplomityötä, mutta päädyimme lopulta erillisiin töihin. Joni kirjoitti Eatech Portablen tietokantojen synkronoinnista vuonna 2014 ja itse aloitin saman projektin käyttöliittymätoteutukseen keskittyvän diplomityön kirjoittamisen vuoden 2015 kesällä.

Luvussa 2 esitellään sovelluksen kokonaisarkkitehtuuri, kerrotaan lähtökohtana olleista reunaehdoista sekä minkälaisia haasteita projektissa on. Luvussa 3 tutustutetaan lukija toteutusympäristöön, johon sovellus on tehty. Esitellään laitteet, käyttöjärjestelmä, sen erityispiirteet sovelluskehittäjän näkökulmasta sekä sovelluksen julkaisutavat Windows 8 -ympäristössä. Lisäksi listataan joitain projektin aikana käytettyjä työkaluja. Luvussa 4 kerrotaan käyttöliittymän arkkitehtuurista ja mitkä ovat olleet syyt tehdyille ratkaisuille. Luvussa 5 paneudutaan toteutuksen yksityiskohtiin tarkemmin ja luvussa 6 arvioidaan projektin onnistumista, jonka jälkeen yhteenveto on luvussa 7.

2. YLEISKUVAUS

Tässä luvussa tarkastellaan toteutettua järjestelmää yleisellä tasolla, jotta lukijalla on parempi ymmärrys kokonaisuudesta, kun tulevissa luvuissa käsitellään aiheita yksityiskohtaisemmin. Lisäksi tuodaan esille toteutetun järjestelmän perimmäinen motivaatio ja mitä haasteita projektissa oli.

2.1 Motivaatio

Sovelluksen kohderyhmältä, eli liikkuvilta myyjiltä, on selkeästi löydettävissä useita tilanteita, joihin tietotekninen ratkaisu tuo helpotusta. Motivaationa sovellukselle ja koko projektille on täyttää kaikki nämä tarpeet ja tarjota yhtenäinen sovellusratkaisu, jolla asiakas pystyy tehostamaan myyntiään ja saamaan siten liiketoiminnallista hyötyä.

Asiakaskäyntien suunnittelua pystymme tehostamaan muun muassa näyttämällä asiakkaiden ostoshistorian kuvaajien ja listauksien avulla. Myyjä voi suunnitella myyntikierroksia karttapohjan päälle visualisoiduilla kohteilla, josta korostuu potentiaaliset ostajat aikaisemman ostokäyttäytymisen perusteella. Tätä näkymää pystyy tehostamaan näyttämällä samalla kartalla liikennetiedot ruuhkien välttämiseksi tai vaihtamalla kartta satelliittikuviin parkkipaikkojen löytämiseksi kohteen läheltä.

Sovellus mahdollistaa asiakastietojen, kuten yhteyshenkilöiden, osoitetietojen, laskutus ja luottotietojen ja esimerkiksi muistiinpanojen näyttämisen ja muokkaamisen asiakaskäyntien yhteydessä, jolloin yrityksen asiakasrekisteri pysyy ajan tasalla. Myös navigointi asiakkaan luokse on mahdollista käyttöjärjestelmän karttasovellus-integraation avulla.

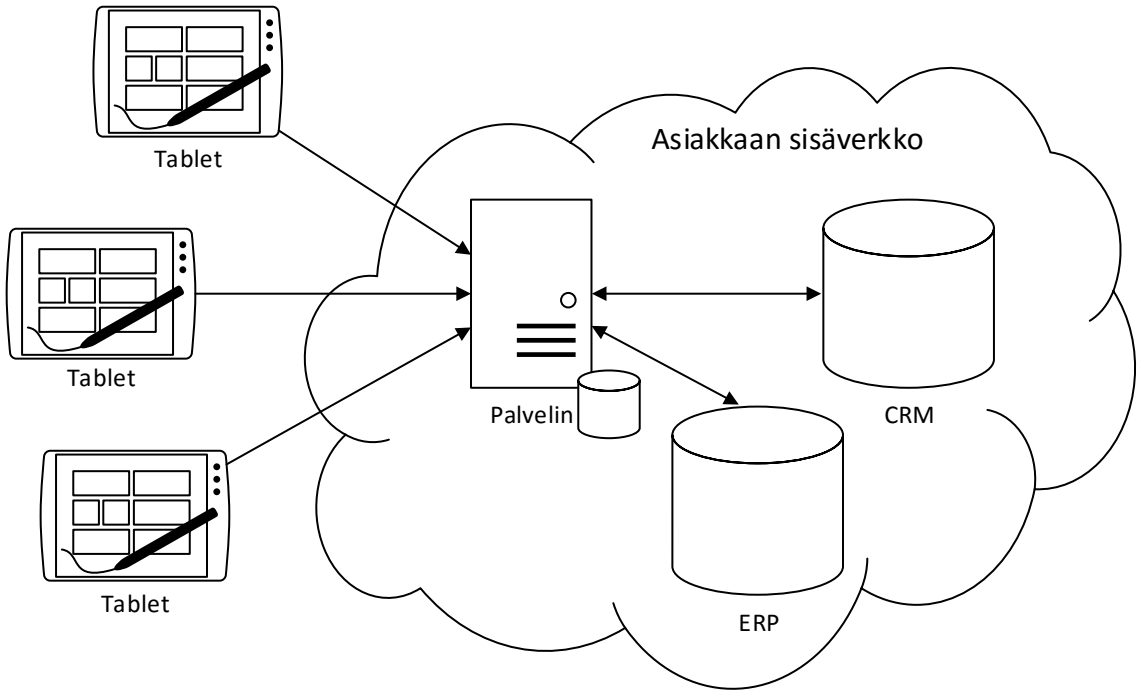
Tuote-esittely tapahtuu ilmaisuvoimaisen listan avulla, jota pystyy ryhmittelemään, suodattamaan, järjestämään ja josta pystyy hakemaan eri tietojen kuten tuotenumeron, -nimen tai kuvauksen perusteella. Listauksessa näkyy kuva tuotteesta olennaisten lisätietojen kuten hinnan, varastosaldon ja tuotenumeron ohella. Tarkempaa tuote-esittelyä varten sovelluksesta löytyy tarkkoja, suurennettavia tuotekuvia lukuisten yksityiskohtaisten tuotetietojen lisäksi.

Tilauksen tekeminen on nopeaa tähän tarkoitukseen suunnitellun käyttöliittymän avulla ja myyjä näkee välittömästi varastosaldon ja saapuvien erien suuruudet sekä toimituspäivämäärät, jotta pystyy myymään juuri oikean määrän tuotteita. Kaikki tiedot ovat käytettävissä offline-tilassa ja mikäli verkkoyhteys on huono, myös tilauksen voi tehdä ilman internet-yhteyttä.

2.2 Yleiskuvaus kokonaisuudesta

Eatech Portable Sales on liikkuvan myyjän tablet-sovellus, joka sisältää laitteeseen ladattuna kaiken myyjän kentällä tarvitseman tiedon myös offline-käyttöä varten. Sovellus on yhteydessä yrityksen omassa toimistoverkossa sijaitsevaan palvelimeen aineistoa synkronoitaessa ja tilauksia lähetettäessä. Palvelimella on tietokanta, johon kaikki järjestelmän tiedot kerätään. Tietokantarakenne on lähes samanlainen tableteilla ja palvelimella synkronoinnin helpottamiseksi. Liitynnät muihin järjestelmiin, kuten toiminnanohjaus- ja asiakkuudenhallintajärjestelmiin, on toteutettu samalla palvelimella ajettavien integraatiopalveluiden avulla. Kuvassa 1 esitellään järjestelmän komponentit.

Laitteet muodostavat automaattisesti VPN-yhteyden yrityksen toimistoverkkoon kommunikoidakseen palvelimen kanssa. VPN-tunnelin ansiosta ei ole tarvetta sallia suoria yhteyksiä julkisesta internetistä yrityksen palvelimelle, jolla säilötään liiketoiminnallisesti arkaluonteisia tietoja.



Kuva 1: Korkean tason havainnekuva Eatech Portablen arkkitehtuurista

2.3 Haasteet

Sovelluksen määrittelyvaiheessa nousi esille haasteita, jotka asettivat vaatimuksia teknologiaratkaisuille ja sovelluksen arkkitehtuurille. Tärkeimmät niistä olivat:

- **Käyttäjälähtöisyys** – Nykypäivänä erityisesti tablet-sovelluksille on ominaista käyttöliittymän viimeistelty toteutus. Enää kuluttajille ei riitä, että sovelluksella pystyy pelkästään tekemään tarvittavat asiat. Sovelluksen tulee olla miellyttävä

käyttää sekä intuitiivinen. Myös yritysmaailmassa sovelluksen visuaalinen ilme ja käyttökokemus vaikuttavat ostopäätöksen tekemisessä. [2]

Tablet-sovellusten tulisi olla siinä määrin yksinkertaisia, että erillisiä ohjeita ei tarvita, vaan käyttäjä osaa käyttää sovellusta heti laitteen käsiin saatuaan. Nopean käyttöliittymän, sulavien animaatioiden ja miellyttävän asettelun sekä värimaailman avulla käyttäjälle tulisi pystyä luomaan positiivinen käyttökokemus, jotta hän ei pelkästään tarvitse, vaan myös haluaa käyttää sovellusta töiden tekemiseen. Mikäli tässä onnistutaan, saa sovellus myös myyntitilanteessa aikaan samat positiiviset reaktiot ja asiakas tekee todennäköisemmin ostopäätöksen.

- **Nopeus** – Sovelluksen lisäarvo asiakkaalle on liiketoiminnan tehostaminen, joten laitteen avulla tulee myyntikäynti pystyä suorittamaan mahdollisimman helposti ja sovellus tulee suunnitella erityisesti tätä tehtävää varten. Tuotteiden selailun, niiden lisäämisen ostoskoriin, lisätietojen katselun ja kaikkien oheistoimintojen pitää toimia ilman minkäänlaista viivettä.

- **Tietomäärät** – Järjestelmän säilömään aineistoon kuuluu tiedot yrityksen asiakkaista, tuotteet mahdollisine tuotekuvineen, tilaushistoria, varastosaldot, yhteyshenkilöt monien muiden tietojen ohella. Tallennetun aineiston määrä riippuu asiakasyrityksestä, mutta yleensä puhutaan tuhansista riveistä esimerkiksi asiakkaiden ja tuotteiden osalta. Nämä kaikki yhdessä kasvattavat järjestelmän sisältämän tiedon määrän kohtuullisen suureksi ja se täytyy pystyä synkronoimaan laitteen ja palvelimen välillä virheettömästi ja nopeasti. Ratkaisu tulee toteuttaa niin, että tietomallia pystyy päivitysten yhteydessä muokkaamaan ilman tuotantokatkoa ja kaikkien laitteiden samanaikaista päivittämistä.

- **Muunneltavuus** – Koska kyseessä on yrityksen oma tuote, jonka hankkijoita ovat yritykset, joilla kaikilla on hieman erilaiset liiketoimintamallit, täytyy sovelluksen olla räätälöitävissä asiakkaan tarpeisiin sopivaksi. Mikäli sovellusta jaetaan yleisen kauppapaikan, eli tässä tapauksessa *Windows Storen* kautta, tulee sovelluksen pystyä lataamaan räätälöintiohjeensa asiakkaan palvelimelta ja mukautumaan niiden mukaan. Sovelluksen arkkitehtuurin tulee olla sellainen, että se mahdollistaa mahdollisimman monipuolisten räätälöintien tekemisen sekä sovelluksen teeman ja värimaailman muokkaamisen asiakasyrityksen näköiseksi. Nämä kaikki pitää sovellusten päivitysten yhteydessä pystyä tekemään niin, että muiden sovellusta käyttävien asiakkaiden toiminta ei häiriinny.

3. TOTEUTUSYMPÄRISTÖ

Tässä luvussa kuvataan korkealla tasolla käyttöympäristö, laitteet ja käyttöjärjestelmä, johon sovellus on toteutettu sekä esitellään muutama keskeinen ulkopuolinen komponentti, jota sovelluksen rakentamisessa on käytetty apuna. Luvussa sivutaan hieman ympäristön asettamia haasteita ja näiden vaikutuksia, jotka vaikuttavat osaltaan sovelluksen arkkitehtuurisissa päätöksissä.

3.1 Tablet käyttöympäristönä

Tablet-tietokoneita on suunniteltu ja kehitetty 1990-luvun alusta asti, mutta vasta Apple onnistui iPad-laitteellaan tekemään läpimurron vuonna 2010 määrittäen samalla alan standardin. Myöhemmin muut valmistajat ovat seuranneet Applen jalanjälkiä ja ympärille on rakentunut kattava laitekirjo. [3]

Microsoft julkaisi jo vuonna 2002 Tablet PC:n, jossa oli kosketusnäyttö ja kynä. Ongelmana tässä laitteessa oli kuitenkin se, että Windowsin käyttöliittymä oli suunniteltu alun perin täysin näppäimistön ja hiiren ehdoilla, eikä sen vuoksi soveltunut kunnolla kosketuslaitteille. Myöskään valmistusmenetelmät eivät olleet tarpeeksi kehittyneitä, jotta riittävän pienien, tehokkaiden, mutta silti pitkällä akunkestolla varustettujen laitteiden valmistaminen olisi ollut mahdollista. [4]

Apple julkaisi yhdeksän vuotta myöhemmin oman tablet-laitteensa, jossa oli tehty paljon rohkeampia ratkaisuja, mitkä erottivat laitteen edeltäjistään. iPadin esikuvana ei pidettykään tietokonetta, mitä monet olettivat, vaan kännykkää. Tämä teki siitä ominaisuuksiltaan erityisen rajoitetun tietokoneisiin nähden, eikä mahdollistanut kaikkea, mihin työpöydillä oltiin totuttu. Toisaalta iPhoneista tuttu iOS-käyttöjärjestelmä oli suunniteltu alkujaankin kosketuskäyttöä varten ja oli sen puolesta myös luonnollinen valinta tablet-laitteisiin. Yksinkertaisesta, helppokäyttöisestä ja ennen kaikkea hyvin viimeistellystä tuotteesta tuli välitön myyntimenestys. Tämän jälkeen Google ja Microsoft ovat tuoneet markkinoille omat kilpailevat alustansa. [3]

3.1.1 Keskeiset piirteet

Tableteille ominaista on todella pitkälle viety helppokäyttöisyys ja laitteiston sirous. Käyttökokemus on noussut tärkeimmäksi tekijäksi ominaisuuksien, tehokkuuden ja yhteensopivuuden ohitse. Laitteita markkinointiinkin aluksi pääsääntöisesti kuluttajille viihdekäyttöön, joskin myöhemmin myös yrityskäyttö on otettu huomioon. [2]

Pienen kokonsa puolesta laitteissa korostuu niiden liikuteltavuus, minkä takia myös akun kesto ja langattomat yhteydet ovat keskeinen osa käyttökokemusta pienen, ideaalitapauksessa minimaalisen fyysisen koon ohella. Laitteista löytyy harvoin muita liitäntöjä kuin USB sekä mahdollinen kuulokeliitäntä. Yrityskäyttöön suunnitelluissa tableteissa saattaa lisäksi olla telakoimismahdollisuus tai liitäntä ulkoiselle näytölle.

Pelkkien tablet-tietokoneiden lisäksi on olemassa myös hybridilaitteita, joissa pystytään tarjoamaan tablettien ja kannettavien hyvät puolet samassa laitteessa. Näissä näyttö on irrotettavissa näppäimistötelakasta, jolloin laite toimii tablettina. Monesti näissä ratkaisuissa näppäimistöön on sijoitettu osa laitteen akustosta, minkä johdosta akunkesto on hieman lyhyempi kuin tablettina käytettynä. Myös jotkin liittimet, oheislaitteet tai esimerkiksi tehokkaampi näytönohjain saattaa olla näppäimistön yhteydessä.

Laitteiden näytöt tukevat lähes aina 5-10 pisteen *multitouch-kosketusta* ja ovat tarkkuudeltaan keskimääräistä PC-näyttöä tarkempia. Pinnoite on yleensä kiiltävä ja taustavalo tarpeeksi kirkas, jotta myös ulkokäyttö on mahdollista. Microsoft on asettanut laitteistovalmistajille vaatimuksia Windows 8 -tablettien näytöille tämän varmistamiseksi. [5]

3.1.2 Yhteydet

Muiden mobiililaitteiden tavoin myös tablettien tiedonsiirtoyhteyksissä keskitytään erityisesti langattomiin ratkaisuihin, mutta muutamien käyttötapauksien johdosta myös muutamia langallisia yhteyksiä löytyy. Tavallisimmat tableteista löytyvät yhteydet ulkomaailmaan ovat seuraavat:

- **3G/4G** – Jotta liikuteltavalla mobiililaitteella voisi päästä kaikkialla nettiin, myös kodin ja työpaikan ulkopuolella, on mobiililaajakaista välttämättömyys. Uusimmissa malleissa *4G*-modeemit ovat korvanneet vanhemmat *3G*-modeemit ja tarjoavat parhaimmillaan lähes langallisten laajakaistojen nopeudet ja vasteajat langattomasti. [6]

- **WLAN** – Paremman virrankulutuksen, siirtonopeuden, vasteaikojen ja käyttökustannusten vuoksi laitteista löytyy myös lähes aina *WLAN*-sovitin, jolla laitteen saa kytkettyä langattomaan lähiverkkoon. [6]

- **Bluetooth** – Lähes kaikista mobiililaitteista nykyisin löytyvä vähävirtainen langaton tiedonsiirtoyhteys on *Bluetooth*. Tätä käytetään erityisesti mobiililaitteiden väliseen tiedonsiirtoon. [6]

- **NFC** – Hyvin läheltä, lähes kosketusetäisyydeltä tapahtuvaan tiedonsiirtoon löytyy monista uusista laitteista *NFC*-tuki. Teknologian avulla laitteet voivat kommunikoida muutaman senttimetrin etäisyydeltä ja siirtää tietoa 106-424 kbit/s no-

peudella. Käyttötarkoituksia löytyy aina lähimaksamisesta NFC-tagien lukemiseen. NFC-tagi on pieni, monesti tarran sisälle upotettu *RFID*-piirin sisältävä siru. Näillä voidaan aktivoida laitteissa esimerkiksi sovelluksia, toimintoja tai avata linkkejä. NFC:tä voidaan käyttää myös esimerkiksi Bluetooth- tai WLAN-yhteyden muodostamisessa, jolloin isompia tietomääriä voidaan siirtää helposti laitteiden välillä viemällä ne toistensa lähelle. [7]

- **GPS** – Käyttöjärjestelmien paikkatietorajapinnat osaavat hyödyntää monia eri lähteitä laitteen sijainnin määrittämiseen. Windows 8:n tapauksessa näitä ovat esimerkiksi WLAN-verkot, IP-osoitetiedot ja laitteen satelliittipaikannuspiirit. Yleisin laitteissa käytettävä satelliittipaikannusmenetelmä on Yhdysvaltojen GPS-järjestelmä, mutta joidenkin laitteiden paikannuspiireistä löytyy tuki myös venäläisten vastaavalle *Glonass*-järjestelmälle. Tulevaisuudessa EU:n *Galileo*-satelliittipaikannusjärjestelmän on lupailtu lisäävän paikannustarkkuutta sekä sen luotettavuutta entisestään. Koska satelliittien löytäminen ja sijaintitiedon saaminen pelkän satelliittijärjestelmän avulla voi joskus kestää minuutteja, löytyy laitteista myös *A-GPS*, eli *Assisted GPS* -tuki. Tällöin muun muassa satelliittien sijaintitiedot ja lentoradat sisältävät satelliittialmanakat ladataan datayhteyden avulla. Monissa laitteissa onkin satelliittipaikannuspiiri ainoastaan, mikäli myös 3G/4G-modeemi löytyy. [8; 9]

- **USB** – Tyypillisin ja useimmissa tableteissa ainut langallinen tiedonsiirtoyhteys on USB. Tätä samaa liitintä on yleensä käytetty laitteen lataamiseen, jolloin porttien määrä saadaan pidettyä pienenä ja laite pysyy yksinkertaisena. Yksi joidenkin Eatech Portablen kanssa käyttämä käyttökohde tälle on USB viivakoodinlukija.

- **Verkkokaapeli** – Lähinnä joistakin hybridilaitteista löytyy enää perinteinen *RJ45-liitin* langallista verkkoyhteyttä varten.

3.1.3 Laskentateho

Pienestä koosta ja pitkästä akunkestostaan huolimatta tabletit ovat suhteellisen tehokkaita peruskäytössä nykypäivänä. Laitteita on saatavilla sekä *x86*- että *ARM*-arkkitehtuurin suorittimilla. Kännyköistä tutut ARM-suorittimet ovat *RISC*-suorittimina käskykannaltaan yksinkertaisempia *CISC*-suorittimiin verrattuna, minkä johdosta ne kuluttavat myös vähemmän virtaa. Niiden haittapuoli kuitenkin on se, että varsinkaan perinteiset Windows työpöytäsovellukset, jotka on käännetty PC:ssä käytetylle x86-käskykannalle, eivät toimi niillä. Windowsista on ARM-pohjaisille tableteille olemassa *Windows RT* -käyttöjärjestelmä, joka vastaa monilta osin Windows 8 -käyttöjärjestelmää, mutta siihen ei kuitenkaan pysty asentamaan esimerkiksi perinteisiä työpöytäsovelluksia. [10]

Useimmat x86-arkkitehtuurin tabletit käyttävät vähän virtaa kuluttavia *Intel Atom* -suorittimia. Joitakin hybridimalleja löytyy, joissa on tehokkaampi *i5*- tai *i7*-suoritin, mutta näiden akunkesto jää Atom- ja ARM-suorittimista. [11]

3.1.4 Varustelu

Pienestä koostaan huolimatta tabletteihin on monesti mahdutettu joitain lisävarusteita lisäämään käyttötapoja ja -mahdollisuuksia:

- **Stylus** – Joissakin tableteissa on *stylus*, eli osoitinkynä käsin kirjoittamista ja piirtämistä varten. Tätä varten laitteessa on monesti oma digitoija sekä kynälle että kosketukselle, joka tulkitsee kosketuksen näyttöpinnalta. Tämän ansiosta pystytään estämään esimerkiksi kämmenellä aiheutetut virhesyötteet, kun osoitinkynä on lähellä ja tunnistettua, millä voimakkuudella kynää painetaan näyttöä vasten. [12]

- **Mikrofoni** – Tableteissa on monesti niin kutsuttu *microphone array*, eli useampi yhdessä toimiva mikrofoni, joiden avulla taustahälinää saadaan vaimennettua ja äänitettyä lähellä kuuluvat äänet laadukkaammin. [13]

- **Kaiuttimet** – Käytännössä kaikista laitteista löytyy sisäänrakennetut stereokaiuttimet äänentoistoon.

- **Kamera** – Vaikka tablet on harvoin pääasiallinen kuvausväline, monista laitteista löytyy etukamera videopuheluita varten ja joskus myös hieman tarkempi takakamera perinteiseen valokuvaukseen.

- **Valoisuusanturi** – Valoisuusanturi on laitteissa yleensä näytön taustavalon tukena, jotta laite osaisi automaattisesti himmentyä pimeässä tilassa ja muuttua kirkkaammaksi auringonpaisteessa. [14]

3.2 Käyttöjärjestelmä ja suoritusympäristö

Sovelluksen toteutuksen ohella myös laitteen käyttöjärjestelmä vaikuttaa olennaisesti kokonaiskäyttökokemukseen. Eatech Portablen tapauksessa asiakkaat ostavat ratkaisun, joka sisältää enemmän kuin pelkän tablet-sovelluksen. Liikkuvan myyjän kaikkien tarpeiden hoitaminen yhdellä laitteella on yksi keskeisimmistä asioista. Sen lisäksi, että myyjä voi käyttää laitetta itse sovellukseen, sähköpostin kirjoittamiseen, netin selailuun tai toimisto-ohjelmien käyttämiseen, käyttää myös Eatech Portable joitain käyttöjärjestelmän sovelluksia esimerkiksi karttojen tai pdf-tiedostojen näyttämiseen. Näiden eri sovellusten käynnistäminen, sulkeminen ja niiden välillä vaihtaminen ovat yksi alustan käyttökokemukseen vaikuttava tekijä.

Suoritusympäristö ja alustan tarjoamat keinot sovelluskehitykseen vaikuttavat myös olennaisesti toteutettavaan sovellukseen. Sovelluksessa pyritään usein käyttämään alustalle ominaisia ratkaisuita ja alusta myös sanelee joitain reunaehtoja sovellukselle.

3.2.1 Windows 8

Windows 8 on Microsoftin ensimmäinen käyttöjärjestelmä, joka on pyritty tekemään käyttöliittymää ja sovelluskehitystä myöten tablet-käyttäjien ehdoilla [15]. Tämä näkyy monina merkittävinä muutoksina aikaisempiin käyttöjärjestelmän versioihin verrattuna. Käynnistä-valikko on korvattu aloitusnäytöllä, josta sovellukset saa käynnistettyä isokokoisista kuvakkeista. Uusia Windows 8 -sovelluksia ei pysty ajamaan enää ikkunatilassa, kuten Windows-ympäristöissä ollaan totuttu, vaan sovellus on koko näytössä tai ruutu on jaettu kahden sovelluksen kesken. Sovellusten sulkeminen ja niiden välillä siirtyminen tapahtuu eleillä, eikä tehtäväpalkki ole näkyvillä, ellei olla perinteisessä työpöytätilassa. Näiden avulla uusien sovellusten käyttäminen ja niiden hallinta on helppoa kosketuskäyttöliittymällä, mutta samalla tämä voi tuntua vanhoihin Windows versioihin tottuneille hiiri- ja näppäimistökäyttäjille kankealta ja alkuun hankalaltakin. [16]

Kun Windows 8 julkistettiin, sille tehtyjä uusia sovelluksia kutsuttiin *Metro-sovelluksiksi* Microsoftin kehittämän *Metro-muotokielen* mukaan, jonka mukaan nekin on ohjeistettu tekemään. Tekijänoikeudellisista syistä johtuen Microsoft joutui kuitenkin luopumaan nimen käytöstä, eikä sen tilalle ole sen jälkeen vakiintunut mitään yksittäistä sanaa. Sovellukset tunnetaan muun muassa Windows 8, Modern tai Windows Kaupan sovelluksina. Tässä työssä niistä käytetään nimitystä Windows 8 -sovellus. [17]

3.2.2 Windows Runtime

Windows Runtime, lyhemmin *WinRT*, on Microsoftin Windows 8:n mukana julkaisema sovelluskehys Windows 8 -sovellusten tekemiseen. WinRT on ainut rajapinta, jolla sovellus pääsee järjestelmän resursseihin kiinni. Kyseessä ei ole .NET Framework eikä sen versio, vaikka nimiavaruudet onkin pyritty pitämään samanlaisina ja ohjelmointi tapahtuu sitä tukevilla kielillä. Taulukossa 1 on listattu WinRT:n keskeisimmät nimiavaruudet. [15]

Taulukko 1: WinRT:n keskeisimmät nimiavaruudet [15]

Nimiavaruus	Selitys
Windows.ApplicationModel	Tarjoaa pääsyn käyttöjärjestelmän ydintoimintoihin ja antaa ajonaikaista tietoa sovelluksesta.
Windows.Data	Sisältää menetelmiä eri tietotyyppien käsittelyyn.
Windows.Devices	Sisältää menetelmiä eri laitteiden käyttämiseen.
Windows.Foundation	Tarjoaa WinRT:n peruskomponentteja asynkronisen suorituksen tueksi perustietotyyppien ohella.

Windows.Globalization	Antaa menetelmät sovelluksen globalisointiin.
Windows.Graphics	Sisältää komponentteja grafiikan tuottamiseen ja esittämiseen.
Windows.Management	Sisältää työkalut sovelluksen hallintaan.
Windows.Media	Sisältää luokkia eri mediatyyppien käsittelyyn, kuten kuvien, videoiden ja äänitteiden.
Windows.Networking	Mahdollistaa tietoliikenneyhteydet.
Windows.Security	Sisältää muun muassa kirjautumiseen ja kryptologiaan liittyviä toiminnallisuuksia.
Windows.Storage	Tarjoaa työkalut muun muassa tiedostojen, kansioiden ja sovelluksen asetusten käsittelyyn.
Windows.System	Sisältää toiminnallisuuksia esimerkiksi sovellusten käynnistämiseen, käyttäjätietojen hakemiseen tai muistinkäytön seuraamiseen.
Windows.UI	Sisältää käyttöliittymäkomponentit.
Windows.Web	Tarjoaa muun muassa *HTTP-client* -toteutuksen.

Windows 8 -sovellukset eivät ole alustariippumattomia, vaan ne käännetään aina kullekin ympäristölle ja suoritinarkkitehtuurille *natiivisovelluksiksi*. Tämä tekee sovelluksista hieman kevyempiä, mikä varsinkin mobiililaitteissa on toivottua. Vaikka kyseessä on natiivi sovelluskehys, tarjoaa WinRT kuitenkin Javan, .NET Frameworkin ja muiden tulkattavien suoritusympäristöjen tavoin roskien keruun, joka perustuu *viitelaskureihin*. [15]

WinRT muuttaa sitä, miten Windows-sovellukset käyttävät laitteiston resursseja. Perinteiset työpöytäsovellukset saavat liki kaikki resurssit vapaaseen käyttöönsä. Windows 8 -sovelluksia sen sijaan suoritetaan tarkoin rajoitetussa *hiekkalaatikossa* [15]. Tälle on monia syitä:

- **Tietoturva** – Sovellus ei oletuksena pääse käsiksi käyttöjärjestelmän tai muiden sovellusten tietoihin, ellei sille anneta siihen erillistä lupaa. Sovellukselle voidaan määrittää taulukon 2 listaamia ominaisuuksia (engl. *capability*), jotka mahdollistavat pääsyn ulkopuolisiin resursseihin. Resursseja voivat olla paikallisten tiedostojärjestelmäsijaintien ohella myös ulkoiset laitteet tai käyttöjärjestelmän palvelut. [18]

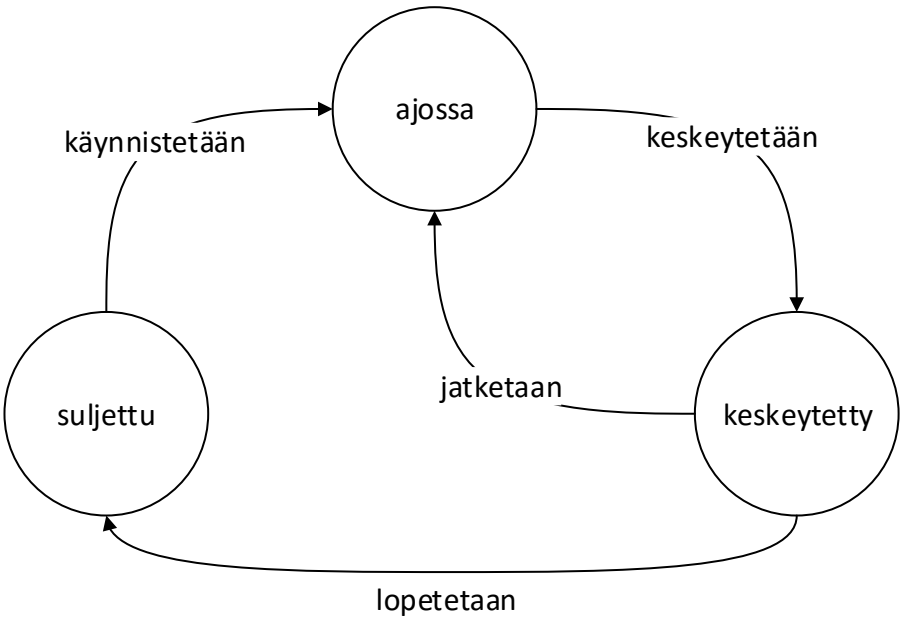
- **Akun kesto** – Erityisesti mobiililaitteissa moniajo voi syödä salakavalasti laitteen akun tyhjäksi hetkessä, mikäli käyttäjän tietämättä taustalla on ajossa raskaita sovelluksia. Windows 8 -sovellusten elinkaari on kuvattu kuvassa 2. Kun käyttäjä vaihtaa toiseen sovellukseen, muuttaa käyttöjärjestelmä taustalle jääneen sovelluksen tilaksi *'keskeytetty'*. Tässä tilassa sovellus ei saa suoritusaikaa, joten myös suorittimen käyttö vähenee ja sen ansiosta myös akku kestää pidempään. [19]

- **Sovellusten pitäminen itsenäisinä** – Microsoft on tarkoittanut Windows 8 -sovellusten olevan omia itsenäisiä kokonaisuuksiaan, jotka käyttäjä saa sellaisenaan asennettua Windows Storesta, ilman että ne vaatisivat muuta konfigurointia tai palveluita järjestelmälle. Tämän johdosta esimerkiksi paikallisesti ajettaville palveluille tai tietokannalle ei pitäisi olla tarvetta ja tämän varmistamiseksi Microsoft on estänyt paikallisten yhteyksien muodostamisen. Yhteyksien muodostaminen voidaan kuitenkin sallia kytkemällä *Loopback Exemption* päälle sovellukselle, joka on asennettu kauppapaikan ohi (*sideloading*). Windows Storen kautta jaeltujen sovellusten kanssa tämä ei kuitenkaan ole mahdollista. [20]

- **Läpinäkyvyys** – Vaatimalla sovellusta listaamaan tarvitsemansa ominaisuudet, Windows Runtime pakottaa sovelluskehittäjän antamaan käyttäjälle samalla myös totuudenmukaisen ja luotettavan kuvan siitä, mitä kaikkea sovellus pystyy tekemään käyttäjän koneelle. [15]

Taulukko 2: Windows 8 sovellusten ominaisuudet [18]

Nimi	Kuvaus
Documents Library	Mahdollistaa pääsyn Omat Tiedostot -kansioon. Sovellus pääsee käsiksi ainoastaan tiedostotyyppeihin, jotka on määritelty sovelluksen manifestissa.
Enterprise Authentication	Sallii Windowsin kirjautumistietojen käyttämisen sovelluksessa sekä kirjautumisen ulkopuolisiin järjestelmiin.
Internet (Client)	Mahdollistaa yhteyksien muodostamisen internettiin.
Internet (Client & Server)	Sallii myös ulkoverkosta saapuvien yhteyksien kuuntelemisen.
Location	Antaa sovellukselle laitteen sijaintitiedot.
Microphone	Sallii mikrofonin käyttämisen.
Music Library	Mahdollistaa pääsyn Musiikki-kansioon.
Pictures Library	Mahdollistaa pääsyn Kuvat-kansioon.
Private Networks (Client & Server)	Sallii yhteydet koti- ja työverkoissa toisiin laitteisiin.
Proximity	Mahdollistaa NFC:n käyttämisen sovelluksessa.
Removable Storage	Mahdollistaa pääsyn massamuistilaitteille. Sovellus pääsee käsiksi ainoastaan tiedostotyyppeihin, jotka on määritelty sovelluksen manifestissa.
Shared User Certificates	Sallii sertifikaattien käyttämisen tunnistautumiseen.
Videos Library	Mahdollistaa pääsyn Videot-kansioon.
Webcam	Sallii webkameran käyttämisen.



Kuva 2: Windows 8 -sovelluksen tilat [19]

Sopimukset (engl. *contract*) ovat Windows 8 -sovellusten kolmas tapa WinRT-rajapinnan ja resurssien ohella kommunikointiin hiekkalaatikon ulkopuolelle. Sovellus voi sopimusten avulla ilmoittaa käyttöjärjestelmälle, että se osaa vastaanottaa esimerkiksi kuvamuotoista tietoa, jolloin käyttöjärjestelmä osaa listata sovelluksen Windows 8:n *Jaa*-listauksessa. Tällöin muista sovelluksista pystyy lähettämään kuvamuotoista aineistoa siihen. Vastaavasti myös muiden sovellusten kanssa pystyy kommunikoimaan niiden tarjoamien, sopimuksilla ilmoitettujen rajapintojen läpi. [15]

Microsoft julkaisi .NET Framework 4.5:n myötä uuden asynkronisen ohjelmointitavan, *Task-based Asynchronous Pattern*:in. WinRT:n myötä tämän käyttäminen on tullut pakolliseksi, sillä rajapinnat tarjoavat monissa tapauksissa ainoastaan asynkronisia versioita toteutuksista. Jotkin säikeen pysäyttävät menetelmät on kokonaan poistettu, kuten `Thread.Sleep`, ja tämän lisäksi käyttöjärjestelmä lopettaa sovelluksen, mikäli säie lukittuu liian pitkäksi aikaa. Tästä johtuen esimerkiksi kaikki verkon yli tapahtuvat kutsut on pakko toteuttaa asynkronisesti tai hieman pidempi viive vastauksissa kaataa sovelluksen. [15]

Asynkronisesta suorituksesta on kuitenkin merkittäviä etuja, joten on ymmärrettävää, että Microsoft on ohjannut sen käyttämiseen. Sovelluksen käyttökokemukseen vaikuttaa merkittävästi käyttöliittymän vastaavuus käyttäjän syötteisiin. Mikäli käyttöliittymä jumittuu edes hetkeksi, aiheuttaa se käyttäjässä negatiivisia reaktioita. Asynkronisella ohjelmointimallilla pystytään pitämään huoli, että käyttöliittymäsäie ei lukkiudu ja näkymä pysyy siten vastaamaan kaiken aikaa käyttäjän syötteisiin ja animaatiot toimivat.

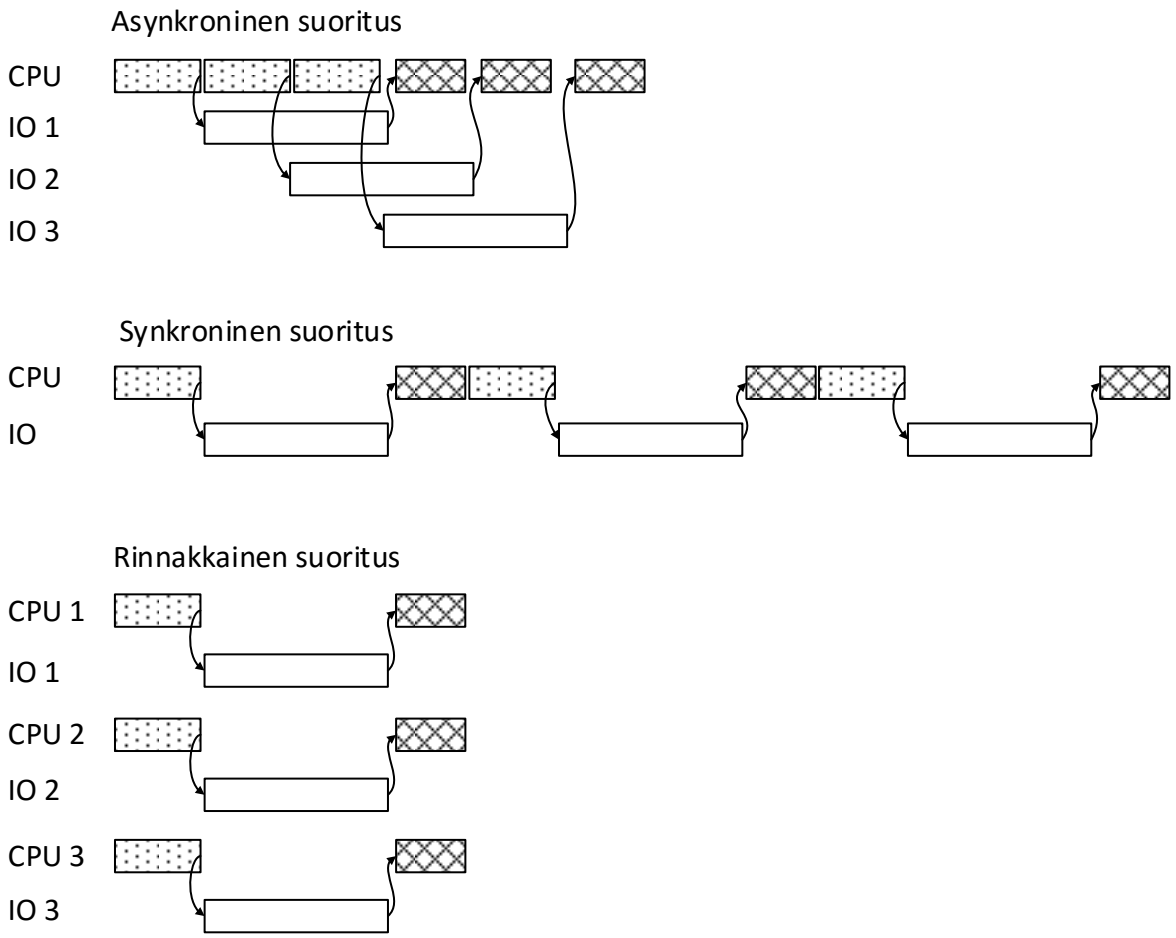
Pelkällä asynkronisella suorituksella ei saa sovellusta kuitenkaan merkittävästi nopeammaksi, vaan sovelluskehittäjän täytyy tunnistaa paikat, joissa saatetaan joutua odottamaan. Näihin käyttöliittymän paikkoihin tulee mahdollisesti lisätä latausanimaatioita, keskeytyspainikkeita tai muita menetelmiä, joilla käyttökokemusta saadaan parannettua.

Task-based Asynchronous Pattern, eli *async/await* -menetelmä helpottaa merkittävästi asynkronista ohjelmointia. Sen avulla asynkronisen koodin kirjoittaminen on mahdollista liki samalla tavalla kuin synkronisen. Listauksessa 1 on pieni esimerkkikoodi asynkronisesta tiedon lataamisesta ja prosessoinnista. `Taskien` eli tehtävien avulla kääntäjä pystyy tekemään koodista eräänlaisen tilakoneen, joka palauttaa suorituksen kutsujalle jokaisen `await`-kutsun yhteydessä, kuten esimerkkilistauksen rivillä 17. [21]

```
 1:  async Task<string> DownloadAndProcessData()
 2:  {
 3:      //Luodaan HttpClient sivun lataamista varten
 4:      HttpClient client = new HttpClient();
 5:
 6:      //GetStringAsync-metodi palauttaa Taskin, jonka kautta paluuarvo on
 7:      // saatavissa myöhemmin. Suoritus alkaa heti, mutta sitä ei odoteta.
 8:      Task<string> data = client.GetStringAsync("http://www.tut.fi");
 9:
10:      //Samalla kun latauksen valmistumista odotetaan, voidaan tehdä muuta.
11:      string prefix = DoSomeProcessing();
12:
13:      //Kun arvoa tarvitaan, saadaan se await-komennon avulla Taskilta. Mikä
14:      // arvo ei ole vielä saatavilla, palautetaan suoritus tämän metodin
15:      // kutsujalle ja jatketaan myöhemmin, ilman että säikeen tarvitsisi
16:      // jäädä odottamaan
17:      string value = prefix + await data;
18:
19:      return value;
20:  }
21:
22:  string DoSomeProcessing()
23:  {
24:      return "data" + ":";
25:  }
```

Listaus 1: Async/await -esimerkki

Asynkroninen suoritus voi vaikuttaa positiivisesti myös sovelluksen nopeuteen erityisesti IO-tehtävien osalla. Kuvassa 3 vertaillaan kolmea eri suoritustapaa toisistaan riippumattomille IO-tehtäville. Asynkronisen suorituksen tapauksessa säie voidaan antaa `await`-kutsun jälkeen välittömästi seuraavan tehtävän käyttöön, jolloin yhdellä säikeellä pystytään suorittamaan lomittain useita IO-kutsuja. Synkronisessa tapauksessa jouduttaisiin aina odottamaan kutsun valmistumista, joten kokonaisaika kasvaisi huomattavasti pidemmäksi. Mikäli tehtäville luotaisiin aina uusi säie, voi suoritus joissain tapauksissa nopeutua, mutta toisaalta myös säikeen luominen ja niiden väliset kontekstin vaihdokset lisäävät kuormitusta ja suurissa määrin tämäkään ei ole tehokasta. Eri säikeiden käyttäminen tuo myös lisäksi rinnakkaisuuden haasteet, ja tulosten saaminen käyttöliittymäsäikeeseen voi lisätä jonkin verran kuormitusta. [22]



Kuva 3: Eri suoritustapojen vertailu [22]

Microsoft on itse toteuttanut WinRT-ohjelmointirajapinnan siten, että kaikki toiminnot, joiden suoritukseen menee yli 50ms, on toteutettu asynkronisesti. Tätä on hyvä pitää ohjesääntönä myös omille sovelluksille. Laskentaintensiiviset tehtävät voi laittaa *Task Parallel Libraryn, TPL*:n avulla `ThreadPoolin` tarjoamaan säikeeseen suoritukseen ja käyttää tätä tehtävää async/awaitin avulla asynkronisesti käyttöliittymäsäikeestä. [15]

3.3 Jakelu ja tuotehallinta

Tablet-sovellukset toivat mukanaan myös uuden tavan asentaa ja hallinnoida niitä. Tämä näkyy helpompana sovellusten jakelemisena ja niiden myynnin sekä toiminnan seuraamisena, mutta samalla myös uusina rajoitteina. Tässä kohdassa käsitellään tarkemmin näistä Windows 8 -alustalla.

3.3.1 Windows Store

Virallinen tapa jakaa sovelluksia Windows 8 -alustoilla on käyttää Windows Store -kauppapaikkaa. Kauppapaikan kautta pystyy selaamaan ja asentamaan uudet Windows 8 -so-

vellukset suoraan laitteelle. Windows Store mahdollistaa myös perinteisten työpöytäsovellusten mainostamisen, mutta niiden asentaminen tai lataaminen ei ole mahdollista kauppapaikan kautta. [23]

Applen ja Googlen kauppapaikkojen tavoin Microsoft pidättää itselleen 30% provision tuotteiden myynnistä Windows Storen kautta. Sovelluksesta voidaan periä maksua myös sovelluksen sisäisillä *In-App*-ostoksilla, joista maksetaan sama 30 prosentin provisio Microsoftille. Kehittäjä voi julkaista enimmillään 10 ilmaissovellusta kauppapaikassa, jotka voivat olla mainosrahoitteisia, aidosti ilmaisia tai esimerkiksi vaatia lisenssikoodin, jonka hankkiminen ja samalla myös veloitus tapahtuu muuta reittiä. [23]

Sovellusten jakaminen Windows Storen kautta antaa kehittäjälle mahdollisuuden sovelluksen toimivuuden, käytön ja myynnin parempaan tarkkailuun sekä tekee sovelluksen päivittämisen helpoksi. Windows Storen kautta on saatavilla niin myynti- ja lataustilastoja kuin myös tietoa kaatumisista ja muista sovelluksen virhetilanteista, jotta kehittäjä voi ennakoivasti korjata ongelmat ennen kuin käyttäjät ilmoittavat näistä suoraan. [24]

Sekä sovellusten lähettäminen että niiden päivittäminen Windows Storeen onnistuu paketoimalla se Visual Studiolla ja lähettämällä paketti nettiportaalin kautta tarkastukseen. Sovellukselle tehdään taulukon 3 mukaiset tarkastukset, jotka läpäistyään se julkaistaan. Windows 8.1 käyttöjärjestelmän myötä sovellukset päivittyvät oletuksena automaattisesti, joten kehittäjä saa helposti ja nopeasti korjauksensa käyttäjien koneille.

Taulukko 3: Julkaisuvaiheet [25]

Vaihe	Kuvaus
Preprocessing	Tässä vaiheessa sovelluspaketin eheys tarkistetaan ja se välitetään eteenpäin tarkempiin testeihin.
Security tests	Sovellus testataan virusten ja haittaohjelmien varalta.
Technical compliance tests	Sovellus testataan automatisoidulla testaustyökalulla, jolla varmistetaan sovelluksen tekninen yhteensopivuus.
Content compliance	Sovelluksen sisältö tarkistetaan.
Release	Sovelluksen julkaisu.
Publishing	Sovellus allekirjoitetaan digitaalisesti.

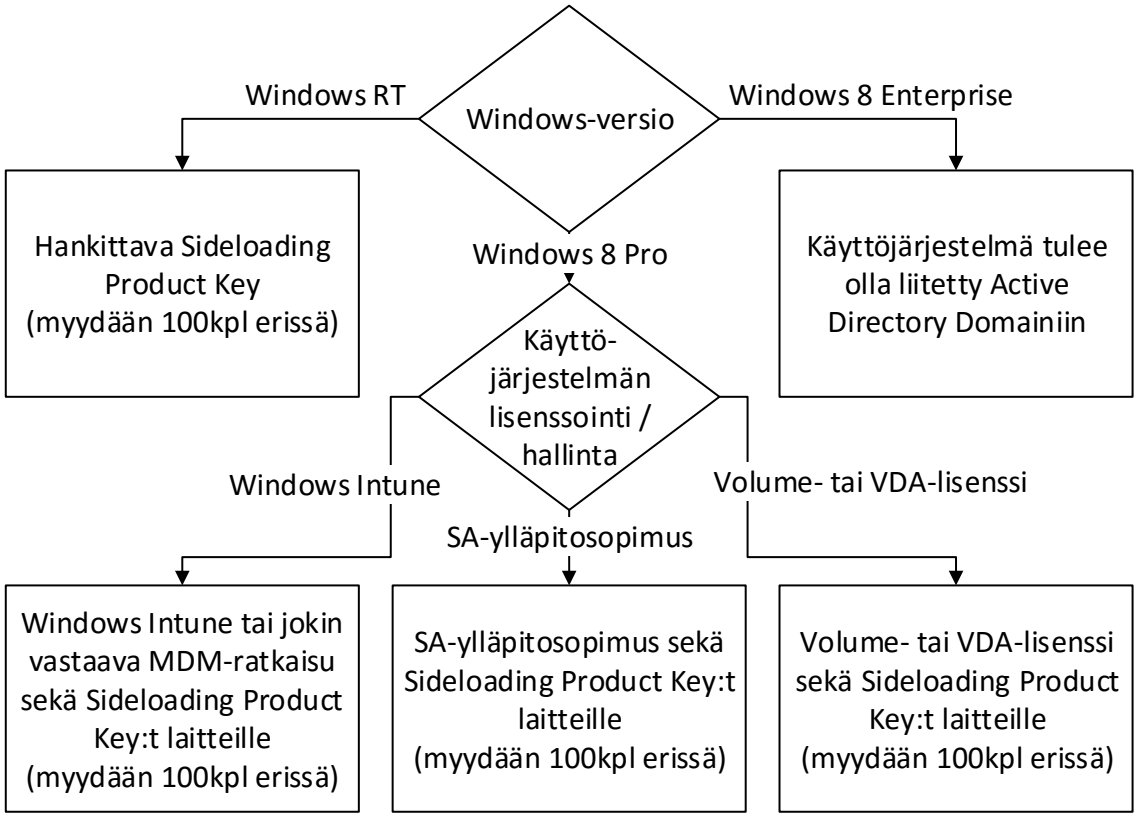
Windows Store on ensisijaisesti toteutettu kuluttajien tarpeisiin, ja monet yritysten vaatimukset jäävät täyttymättä. Yrityssovellusten tapauksessa ei välttämättä ole mielekästä, että sovellus on kaikkien nähtävillä ja ladattavissa julkisessa kauppapaikassa. Yritysten välisessä sovelluskehityksessä maksamista ei välttämättä haluta tehdä Windows Storen kautta, vaan lisensointi täytyy monesti rakentaa itse, jotta projektin voi laskuttaa asiakkaalta suoraan. Myös julkaisemiseen menevä parin päivän viive tarkastusten johdosta voi olla liian pitkä, mikäli kyseessä on tuotantokriittinen sovellus ja korjauksia tarvitaan nopeammalla vasteajalla. [23]

Windows Store rajoittaa joiltain osin sovelluksen ominaisuuksia ja estettyjen joukossa on muun muassa *Loopback Exemption*:in tekeminen. Käytännössä Windows Storen kautta ei siis voi jaella sovelluksia, jotka ottaisivat yhteyden samalla koneella suoritettavaan palveluun. Myös esimerkiksi taulukon 2 *Documents Library* -ominaisuutta on rajoitettu ja sitä käyttäviä sovelluksia voi julkaista Windows Storessa ainoastaan yritykset. [26]

Mikäli kyseessä on *Eatech Portablen* kaltainen liiketoimintasovellus, myös räätälöinti voi osoittautua haasteelliseksi. Windows Storen kautta jaeltaessa kaikki asiakkaat käyttävät samaa sovellusta, jota täytyy monesti muokata hieman asiakkaiden tarpeiden mukaan. Tämä tekee muutosten tekemisen työläämmäksi ja kasvattaa testauksen merkitystä.

3.3.2 Sideloading

Sovelluksen asentamista laitteen pääasiallisen jakelukanavan ohi kutsutaan *sideloading*:ksi. Monesti edellä esitetyt rajoitteet yrityssovellusten jakelussa Windows Storen kautta pakottavat yritykset tarjoamaan sovelluksiaan käyttäjilleen kauppapaikan ohi. Kaavio 4 esittelee sideloading-vaatimukset Windows 8 -käyttöjärjestelmässä. Windows 8.1 Update 1 -käyttöjärjestelmäpäivitys toi mukanaan pienen muutoksen tähän, jonka jälkeen kaikki *Active Directory Domainiin* liitetyt Windows 8.1 Pro -käyttöjärjestelmät sallivat sovellusten asentamisen Windows Storen ohi. Tällöin jakelu onnistuu Pro-versiolla vastaavalla tavalla kuin Windowsin Enterprise -versioilla. [27; 28]



Kuva 4: Sideloading-vaihtoehdot [27]

Monissa yrityksissä, joissa on käytössä Active Directory Domain -hakemistopalvelu, on myös usein jo valmiiksi asennettuna tai mahdollisuus hankkia Enterprise-versio käyttöjärjestelmästä. Tällöin kuvassa 4 listatut vaatimukset täyttyvät. Niiden lisäksi täytyy Active Directoryn ryhmäkäytännöistä sallia sovellusten asentaminen Windows Storen ohi ja laitteen tulee luottaa sovelluksen sertifikaatin myöntäjään [27].

Varsinaisen sovelluksen asentaminen, päivittäminen ja poistaminen tapahtuu Windows PowerShell -konsolin komennoilla `Add-AppxPasckage` ja `Remove-AppxPackage`. [27]

3.4 Toteutustekniikat

Sovellusta tehdessä tarvittiin useita komponentteja ja menetelmiä, joita ei WinRT tarjonnut valmiina. Tässä kohdassa tarkastellaan keskeisimpiä niistä.

3.4.1 WinRT XAML Toolkit

Koska Microsoft joutui julkaisemaan Windows 8:n sekä sen mukana tulleen Windows Runtime -kehitysympäristön kiireellä tablet-markkinoiden kovan kilpailutilanteen vuoksi, on ymmärrettävää, että Microsoftin ensimmäinen tablet-sovelluksien kehittämiseen erikoistunut sovelluskehys on monilta osin puutteellinen. Monet Windows Runtimen edeltäjänä pidetyn *WPF:n* ominaisuuksista ja komponenteista puuttuvat tai eivät ole tarpeeksi räätälöitäviä eri käyttötarkoituksiin. [29]

Microsoftin käyttöliittymäkirjastoille on tavannut löytyä aina isompi avoimen lähdekoodin komponenttikirjasto, joka paikkaa käyttöliittymäkirjaston puutteita: *Extended WPF Toolkit*, *Silverlight Toolkit* ja nyt myös *WinRT XAML Toolkit*.

WinRT XAML Toolkit on avoimen lähdekoodin käyttöliittymäkirjasto, joka sisältää osittain WinRT:lle sovitettuja Silverlight Toolkitin komponentteja, mutta myös uusia erityisesti kosketuskäyttöliittymiä varten suunniteltuja kontrolleja [30].

3.4.2 Bing Maps

Sovelluksen ilmaisuvoimaa ja käytettävyyttä haluttiin lisätä integroimalla siihen tuki kartoille. Selvityksen perusteella yksi monipuolisimmista karttakontrolleista WinRT-alustalle on Microsoftin toteuttama *Bing Maps*. Se käyttää kartta-aineistonaan Nokian hankkimia Navteqin karttoja ja tarjoaa lisäksi muun muassa satelliittikuvat sekä reaaliaikaiset liikennetiedot. [31]

Kontrolli tarjoaa monipuoliset laajennusmahdollisuudet. Sen näkymää voi muokata, suurennusta ja kohdetta voi siirtää ohjelmallisesti, ja esimerkiksi paikkamerkkejä piirtää omien tarpeiden mukaan. Tämän avulla pystytään toteuttamaan esimerkiksi asiakkaan

valinta kartalta niin että eri asiakkaat korostetaan eri väreillä jonkin arvon perusteella. Samoin on myös mahdollista kohdistaa kartta valittuun asiakkaaseen ja näyttää sen sijainti kartalla. [31]

Mikäli laitteesta löytyy GPS-vastaanotin, myös käyttäjän sijainti saadaan helposti näkymään kartalla tarkkuustiedon ohella. Ilman GPS-vastaanotintakin voidaan halutessa käyttää epämääräisempää, esimerkiksi WLAN-tukiasematietojen perusteella laskettua sijaintia.

3.4.3 MVVM Light Toolkit

Käyttöliittymän arkkitehtuuri on suunniteltu *MVVM-suunnittelumallin* mukaisesti. Kyseinen malli esitellään tarkemmin kohdassa 4.3. Tämän toteuttamisen apuna on sovelluksessa käytetty MVVM Light Toolkittia, joka tarjoaa kevyen rungon sovellusten rakentamiseen ja sisältää välttämättömimmät komponentit arkkitehtuurin toteuttamiseen.

3.4.4 SqLite

Microsoftin vision mukaisesti Windows 8 -sovellukset kommunikoivat ensisijaisesti pilven ja muiden Internetissä sijaitsevien palvelimien kanssa, jotka voivat käyttää tietokantoja muun muassa sovelluksen näyttämän datan tai käyttäjän tietojen tallentamiseen. Koska Windows 8 -sovelluksien on tarkoitettu olevan itsenäisiä kokonaisuuksia ja siten täysin riippumattomia muista laitteeseen asennetuista palveluista tai sovelluksista, ei Microsoft ole nähnyt myöskään tarpeelliseksi ottaa .NET Frameworkista tuttua *System.Data*-nimiavaruutta mukaan Windows Runtimeen. Tämä sisältäisi tarvittavat komponentit muun muassa tietokantayhtyeyksien luomiseen ja siten mahdollistaisi ulkoisten tietokantojen käyttämiseen.

Koska offline-tuki on yksi Eatech Portablen keskeisimmistä ominaisuuksista ja sovellus käsittelee paljon dataa, jolla on monimutkainen tietomalli, on paikallisen tietokannan käyttäminen kuitenkin välttämätöntä. Yksi ja sovelluksen toteutushetkellä ainut vaihtoehto tämän toteuttamiseen oli kolmannen osapuolen toteuttama *SqLite-kirjasto*.

SqLite on kevyt tiedostopohjainen relaatiotietokanta, jossa erillisen tietokannanhallintajärjestelmän sijasta tietokantaa käsitellään suoraan sovelluksesta erillisen kirjaston avulla. Tietokannan eheys taataan lukitsemalla tiedosto transaktioiden ajaksi. [32]

Eatech Portablessa on käytetty WinRT-ympäristöillekin saatavaa SqLite for Windows Runtime -PCL-kirjastoa, joka tarjoaa rajapinnan SqLite-tietokantojen käyttämiseen.

3.4.5 SyncFramework

Eatech Portable Sales säilöö keskitetysti asiakkaan palvelimella tiedot muun muassa heidän asiakkaista, tuotteista ja tilauksista. Koska sovelluksen on oltava käytettävissä myös offline-tilassa, nämä samat tiedot, käytännössä suurin osa koko tietokannasta, täytyy pystyä synkronoimaan laitteisiin.

Koska sovellusta käytetään monesti hitaiden mobiiliyhteyksien yli, synkronoinnin tulee olla tehokasta, virhesietoista, sovelluksen tulee siirtää ainoastaan tarvittava data ja hallita mahdolliset konfliktit molemmissa päissä. Muun muassa nämä monien muiden seikkojen ohella tekevät tietokantojen synkronoimisesta niin monimutkaisen kokonaisuuden, että päädyimme käyttämään sovelluksessa valmista Microsoftin toteuttamaa Sync Framework -kirjastoa.

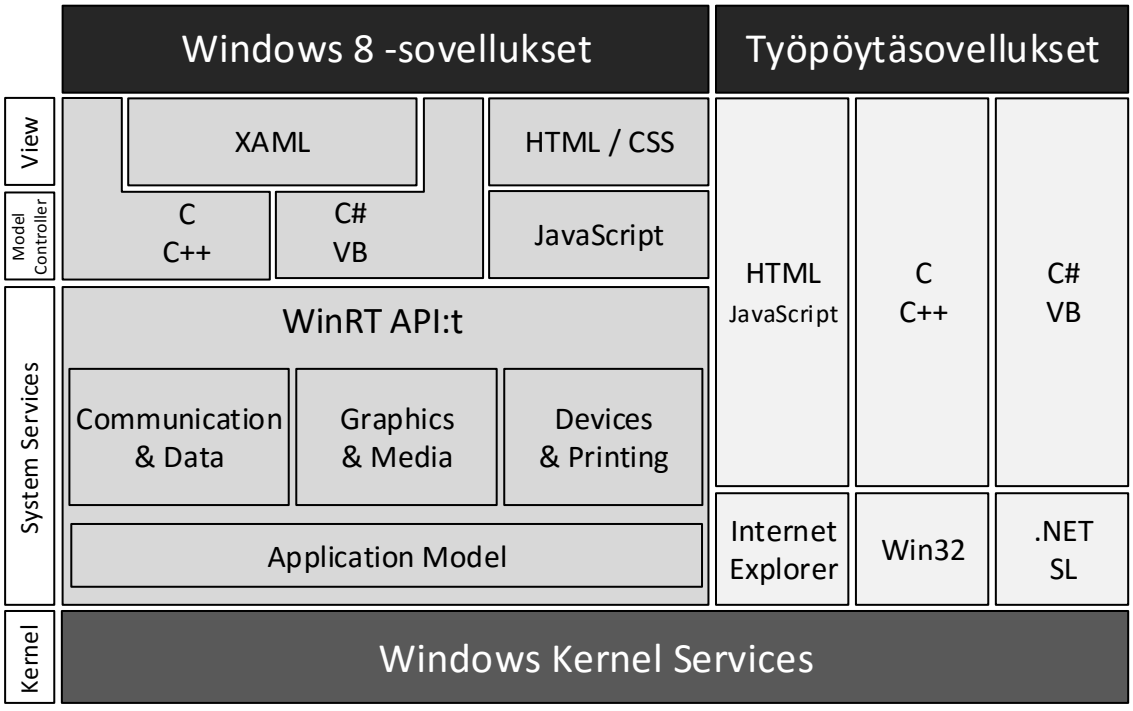
Sync Framework tukee monenlaisia synkronointitopologioita, tietokantajärjestelmiä ja käyttöympäristöjä. Tuettuna ovat sekä palvelimen .NET Frameworkilla toteutettu Microsoft SQL Serveriä käyttävä *Windows Service*, kuin myös laitteiden päässä Windows Runtimella toteutettu SqLitea käyttävä tablet-sovellus. [33]

4. KÄYTTÖLIITTYMÄN TOTEUTUS

Tässä luvussa käsitellään tarkemmin tablet-sovelluksen käyttöliittymän toteutusta ja siinä käytettyjä menetelmiä. Esitellään käytetyt arkkitehtuuriratkaisut sekä joitain teknisiä yksityiskohtia, jotka ovat olennaisia sovelluksen toiminnan kannalta.

4.1 Yleiskuvaus

Kuvassa 5 on Windows 8:n tarjoamat menetelmät sovelluskehitykseen. Kuten alakohdassa 3.2.2 todettiin, kaikki kutsut käyttöjärjestelmälle menevät WinRT-API:n kautta. Windows 8 tarjoaa kolme tapaa toteuttaa Windows 8 -sovelluksia. Mikäli web-teknologiat ovat tuttuja, voi HTML+CSS yhdistelmä olla luonteva lähestymistapa käyttöliittymän toteuttamiseen. Tällöin sovelluslogiikka kirjoitetaan JavaScriptillä. Kaksi muuta tapaa ovat C/C++ tai .NET kielet C# ja VB. Molempien tapauksessa näkymät kuvataan XAML-kuvauskielellä, joka on hyvin lähellä WPF:n tai Silverlightin XAML-koodia. [15]



Kuva 5: Windows 8 kehitysalusta [15]

Kaikissa tapauksissa käytettävissä on koko WinRT-sovelluskehys kaikkine toimintoineen, ja sovelluksen pystyisi tekemään kaikilla niistä. Eatechilla on kuitenkin vahva ja pitkä osaaminen .NET-pohjaisesta kehityksestä C#-kielellä, joten sen pohjalta oli luontevinta valita se tämän projektin toteutuskieleksi.

Sovelluksen käyttöliittymä on pyritty pitämään ensisijaisesti mahdollisimman yksinkertaisena ja minimoimaan eri näkymien määrä. Koska WinRT tuo mukanaan panoraamanäkymän, joka soveltuu hyvin Windows 8 -sovellusten ilmeeseen, oli sen käyttö luontevaa tässä.

Sovelluksessa on kolme eri tilaa:

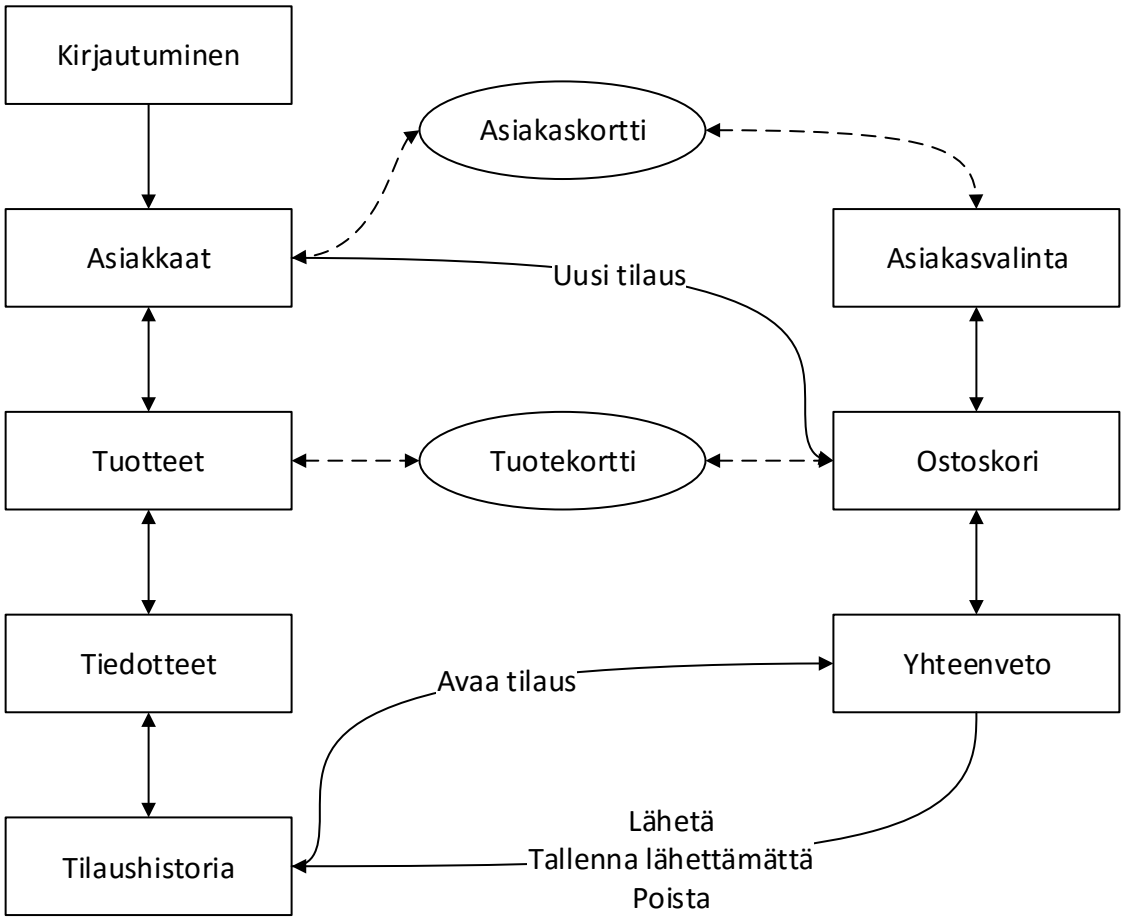
1. Kirjautuminen.
2. Selailutila, josta näkee asiakkaat, tuotteet, tiedotteet ja vanhan tilaukset.
3. Ostostila, jossa voi lisätä tilaukselle tuotteita ja lähettää tilauksen.

Kuva 6 näyttää nämä kolme eri tilaa päällekkäin. Kuvasta näkee, kuinka sivusuuntainen rullaus mahdollistaa tilan eri näkymien välillä siirtymisen. Ylimmässä on kirjautumisnäkymä, josta onnistuneen kirjautumisen jälkeen siirrytään toisella rivillä näkyvään selailutilaan. Selailutila on sivuttaissuunnassa leveä näkymä, jossa pystyy siirtymään sormella näkymää liikuttamalla. Näkymä lukkiutuu aina sivujen kohdalle, joita selailunäkymän tapauksessa on neljä. Seuraava sivu ilmoittaa käyttäjälle olemassa olostaan ”kurkkaamalla” hieman ruudun oikeasta reunasta. Mikäli käyttää aloittaa uuden tilauksen tekemisen tai avaa vanhan, siirtyy hän kuvan viimeisellä rivillä näkyvään ostostilaan. Liitteessä A on tarkemmat kuvat kustakin näkymästä.

Kuva 6: Näkymäkuva

Kuvan 7 näkymäkartasta näkyy kaikki mahdolliset siirtymät sovelluksen sisällä. Alakohdassa 5.2 esiteltäviin sivupalkkeihin pystyy sekä selailu- että ostosnäkymästä tuomaan

asiakastietokortin tai tuotetietokortin näkyviin. Nämä näyttävät valitusta asiakkaasta/tuotteesta tarkempia tietoja ja mahdollistavat tarkemman porautumisen tietoon.



Kuva 7: Siirtymäkaavio

Sovellus sisältää lisäksi synkronointi-painikkeen ruudun oikeassa yläreunassa käyttäjän ollessa selailunäkymässä. Painike on aktiivinen ainoastaan, kun käyttäjällä on yhteys palvelimelle ja synkronointi on tehtävissä. Kun synkronointi on käynnissä, estää käyttöliittymä tilauksien avaamisen tai luomisen. Tämä rajoite luotiin varmistamaan, että kesken tilauksen teon tilaukseen liittyvien tuotteiden mahdollinen häviäminen tai muut muutokset eivät sotke käyttöä. Mikäli edellisestä synkronoinnista ei ole kulunut liian kauaa, itse synkronointiin ei kuitenkaan mene hitaillakaan yhteyksillä montaa minuuttia, joten tämä rajoite ei juuri haittaa. Painikkeeseen on tarkoitus tulevaisuudessa laittaa näkyviin myös arvioitu synkronoinnin kesto ja jäljellä oleva aika, mutta tällä hetkellä tätä ominaisuutta ei ole.

4.2 Suunnittelu

Koska hyvä käyttökokemus ja käyttöliittymän tehokkuus ovat sovelluksen keskeisimpiä tavoitteita, päätettiin erityisesti käytettävyyden suunnitteluun käyttää resursseja. Vaikka asiakkaalla oli Windows Mobile -alustalle toteutettu vanha järjestelmä käytössä, emme käyttäneet tämän toteutusta mallina, jotta emme vahingossa jumittuisi uudella alustalla sille soveltumattomien vanhojen ratkaisuiden käyttöön. Käyttöliittymä on suunniteltu uudelleen käyttäjäkeskeisen suunnittelun menetelmin. Pyrimme tunnistamaan keskeisimmät tarpeet, joita sovelluksen käyttäjillä on ja toteuttamaan nämä mahdollisimman helpoiksi. Tätä tehdessä pitämään kuitenkin mielessä kaikki muut käyttöskenaariot ja toiminnallisuudet, jotta myös ne saadaan sovellukseen mukaan saumattomasti.

Kohdassa 2.3 listattuja haasteita silmällä pitäen tunnistimme sovellukselle seuraavat keskeiset vaatimukset:

4. myyntitilauksen helppo tekeminen,
5. tuotteiden esittely asiakkaalle,
6. tuki offline-työskentelylle,
7. nykyisen varastosaldon ja saapuvien erien näkeminen, sekä
8. nopea asiakkaiden ja tuotteiden löytäminen.

Näiden lisäksi myös seuraavat tarpeet havaittiin:

9. kartan käyttö asiakkaan valitsemisessa ja asiakastiedoissa,
10. asiakastietojen päivitys,
11. kuukausittainen euromääräinen myynti asiakkaille listana ja diagrammina,
12. navigointisovelluksen integrointi,
13. tuotteiden ja asiakkaiden ryhmittely,
14. tuki viivakoodinlukijalle,
15. kirjautuminen,
16. tehokas synkronointi,
17. tilauksen automaattitallennus, keskeneräisenä tallentaminen ja jatkaminen,
18. tiedotteet, sekä
19. hinnan, määrän ja alennuksen syöttäminen käsin.

Käyttöliittymästä tehtiin useita hahmotelmia, joista nykyinen kahteen eri tilaan jakautuva rullattava panoraamanäkymä tarjosi kaikki ominaisuudet yksinkertaisimmalla tavalla. Sen avulla käyttäjä hahmottaa heti, että oikeasta reunasta ”kurkistaa” seuraava näkymä ja rullaus tapahtuu intuitiivisesti. Kun käyttäjä on tällä tavalla selannut koko näkymän vasemmasta oikeaan reunaan, on hänellä samalla kokonaiskuva kaikesta, mitä sovelluksesta löytyy. Synkronointi-painike on kaiken aikaa näkyvillä ja käyttäjä näkee painikkeesta synkronoinnin tilan. Sekä tuote- että asiakaskohtaiset toiminnot tulevat esille koskettamalla haluttua asiakasta/tuotetta.

Asiakkaiden ja tuotteiden löytämiseen on toteutettu useita menetelmiä, joiden yhteiskäyttö tekee löytämisestä nopeaa. Sekä asiakkaat, että tuotteet on ryhmitelty, ja semanttisen näkymän avulla tuote- ja ryhmittelynäkymän välillä pystyy siirtymään nopeasti sormieleillä. Ryhmät on värikoodattu visuaalisen hahmottamisen parantamiseksi. Lisäksi tuloksia voidaan suodattaa tekstihaulla, joka hakee useista eri kentistä tietoa ja joka päivittyy jokaisella näppäinpainalluksella. Lisäksi on mahdollista näyttää esimerkiksi myyjän omat asiakkaat tai ainoastaan varastossa olevat tuotteet. Nämä kaikki yhdistettynä nopeaan listanäkymään, jolla pystyy rullaamaan monia satoja tuloksia sekunnissa ilman minkäänlaista latausviivettä antavat sulavan käyttökokemuksen.

Jatkokehityksen aikana havaitsimme, että ostostilassa pelkkä tuotteen koskettaminen sen lisäämiseksi ostoskoriin oli liian hidas siinä tapauksessa, että tuotteita haluttaisiinkin enemmän. Tämä ratkaistiin tarjoamalla lisäksi mahdollisuus syöttää lukumäärä käsin tai pitämällä sormea hetki painikkeen päällä, jolloin lukumäärä kasvaa kiihtyvällä tahdilla.

Kaikkien näiden toimintojen avulla saimme tuotteen ydintoiminnon, tilauksen tekemisen niin helpoksi, että tilauksen tekeminen vaati vain kolme painallusta tuotteiden valitsemisen lisäksi.

4.3 Arkkitehtuuri

Tässä kohdassa tarkastellaan sovelluksen arkkitehtuuria ja siinä käytettyjä teknisiä ratkaisuja tarkemmin. Tuon esiin myös joitain syitä, miksi näihin on päädytty.

4.3.1 Kokonaisarkkitehtuuri

Sovelluksen arkkitehtuuri mukailee päällisin puolin palvelukeskeistä arkkitehtuuria (*SOA – Service Oriented Architecture*). Palvelukeskeisessä arkkitehtuurissa eri komponentit on toteutettu itsenäisinä kokonaisuuksina, jotka kommunikoivat toistensa kanssa erikseen määriteltyjen rajapintojen välityksellä. Tiedon välitykseen käytetään standardimenetelmiä ja tietomallit eivät sisällä alustakohtaisia rakenteita. [34]

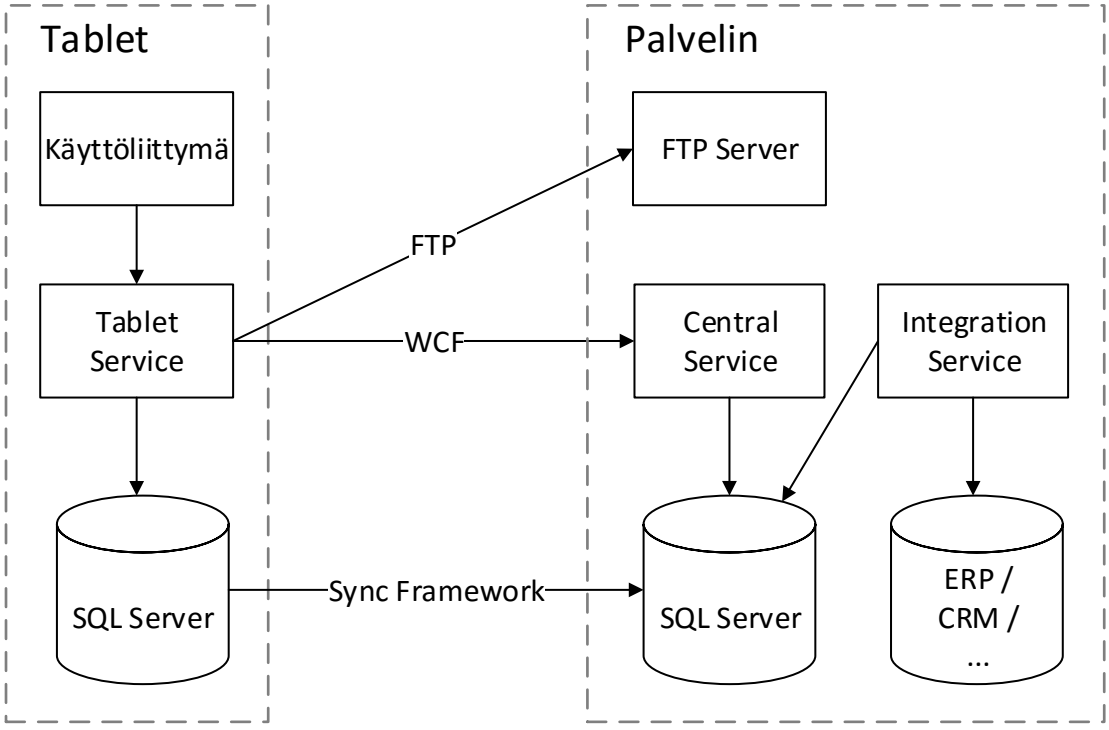
Eatech Portablessa käytetään tiedonsiirtoon *Windows Communication Foundationia (WCF)*. WCF-sovelluskehyksessä palvelut kommunikoivat keskenään rajapintojen välityksellä. Sovelluskehyksessä palvelun käyttäjä luo `ChannelFactory`-luokan avulla edustajan (engl. proxy), joka välittää palvelun pyynnöt rajapinnan kautta kutsuttavalle osapuolelle. Yhteisessä luokkakirjastossa on määritelty myös tiedonsiirrossa käytetyt tietotyypit, *DTO*:t (*Data Transfer Object*). Siirtotieksi Eatech Portablessa on WCF:n XML-konfiguraatiotiedostoissa määritelty *Web Service* –yhteys, eli HTTP-protokollan päälle toteutettu standardi XML-pohjainen viestinvälitys menetelmä. [35]

Tietokannan käsittelyn helpottamiseksi sovelluksessa käytetään olio-relaatiokuvausta (*ORM – Object-Relational Mapping*). Microsoftin *ORM*-ratkaisun, *Entity Frameworkin*

avulla tietokannasta luodaan oliopohjainen esitys. Tietokannan taulujen elementtejä kutsutaan *entiteeteiksi*. Sovelluksessa *entiteetit* ja DTO:t ovat rakenteeltaan ja jäsenten nimeämiseltään todella lähellä toisiaan, joten *AutoMapper*-kirjaston avulla pystymme automaattisesti tekemään muunnokset molempiin suuntiin. [36]

Suuren tietomäärän ja laitteiden hitauden vuoksi jouduimme kuitenkin hieman joustamaan arkkitehtuurista käyttöliittymän päässä. Käyttöliittymässä joissain paikoissa tietoa näytetään sovelluksessa suoraan DTO-olioista ilman niiden muuntamista näkymäkerroksen omiin luokkiin.

Sovelluksen mittavan datamäärän vuoksi tietokannan käyttäminen myös tablet-päässä on välttämättömyys. Sovelluskehityksen alkuvaiheessa ei ollut kuitenkaan olemassa vielä mitään menetelmää muodostaa tietokantayhteyksiä WinRT-sovelluksesta tietokantoihin, eikä erillisiä kirjastoja tiedostopohjaisten tietokantojen käyttämiseen. Ainoaksi vaihtoehdoksi jäi erillisen palvelun asentaminen samalle laitteelle, joka vastasi tietokantayhteydestä ja tiedonsiirrosta asiakkaan palvelimen kanssa. Tämä johti kuvan 8 mukaiseen arkkitehtuuriin.

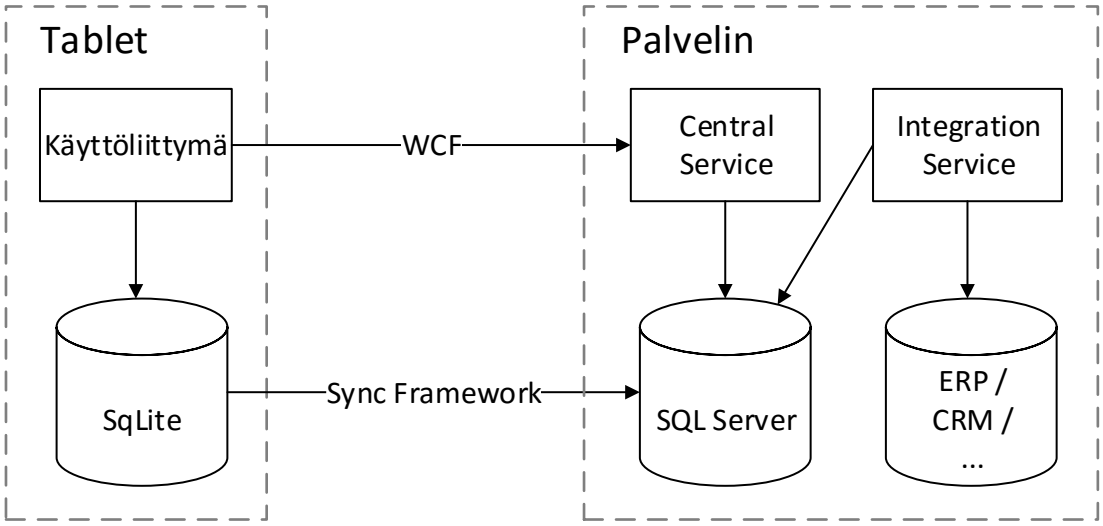


Kuva 8: Eatech Portablen vanha arkkitehtuuri

Koska käyttöliittymästä muodostetaan yhteys samalla koneella ajossa olevaan Windows Serviceen, vaatisi se Loopback Exemptionin tekemistä kuten alakohdassa 3.2.2 todettiin, joten ainoa tapa saada sovellus käyttäjän koneelle olisi sen asentaminen Windows Storen ohi, eli *Sideloading*. Tämä kaikki teki arkkitehtuurista valitettavasti myös samalla todella monimutkaisen asennusten kannalta. Jotta sovellus saataisiin toimimaan käyttäjän laitteella tulisi:

- liittää tietokone yrityksen Domainiin, ellei ole liitetty jo,
- asentaa Microsoft SQL Server,
- ajaa tietokantaskriptit,
- asentaa Tablet Service,
- määritellä ryhmäkäytännöistä sideloading sallituksi,
- asentaa sovelluksen allekirjoittaneen myöntäjän sertifikaatti luotetuksi laitteella,
- asentaa Käyttöliittymä,
- tehdä Loopback Exemption,
- asentaa Sync Framework, sekä
- luoda sovellustunnus, määritellä palvelu käyttämään sitä ja antaa sille tarvittavat oikeudet tiedostojärjestelmään ja tietokantaan.

Vaikka nämä kaikki vaiheet automatisoisi mahdollisimman pitkälle, olisi asennusprosessi silti työläs ja virhealtis. Lisäksi se vaatii, että asiakkaalta löytyy Windows Domain -ympäristö. Kaikkien asiakkaiden kanssa tämä ei välttämättä toteudu, minkä johdosta SqLiten WinRT-version julkaisemisen jälkeen muutimme sovelluksen arkkitehtuurin kuvan 9 kaltaiseksi.



Kuva 9: Eatech Portablen uusi arkkitehtuuri

Tässä arkkitehtuurissa tablet-laitteella ei suoriteta lainkaan Windows Serviceä, vaan Windows 8 -sovellus sisältää kaiken tarvittavan toimiakseen. Tietokantana on SqLite, jonka sovellus luo ja alustaa itse käynnistyessään, mikäli sitä ei löydy. Sync Framework on integroitu tablet-sovellukseen ja se kykenee synkronoimaan SqLite-tietokannan palvelimen Microsoft SQL Server -tietokannan kanssa. Käyttöliittymä on suoraan yhteydessä palvelimen Central Serviceen ja nyt tätä putkea pitkin menee myös aikaisemmin FTP:n kautta siirretyt isot kuvatiedostot ja muu suurikokoinen materiaali.

Asennusprosessi yksinkertaistui myös merkittävästi. Nyt kaikki voivat ladata sovelluksen Windows Storen kautta ja konfiguroida sen ottamaan yhteyttä oman yrityksen Eatech Portable -palvelimeen. Mikäli asiakkaalle räätälöidään oma versio sovelluksesta, jolloin

se jaellaan Windows Storen ohi Sideloading-menetelmällä, täytyy lisäksi vain huolehtia sen asettamista vaatimuksista.

Windows Storeen julkaisun myötä sovelluksen markkinoinnin yhteydessä voidaan mahdollisia asiakkaita myös kannustaa lataamaan sovellus kauppapaikasta ja kokeilemaan sitä itse. Demo-käyttöä varten sovellukseen integroitiin pieni määrä esimerkkiaineistoa, joka tulee näkyville, mikäli sovellukseen ei määritetä palvelimen osoitetta.

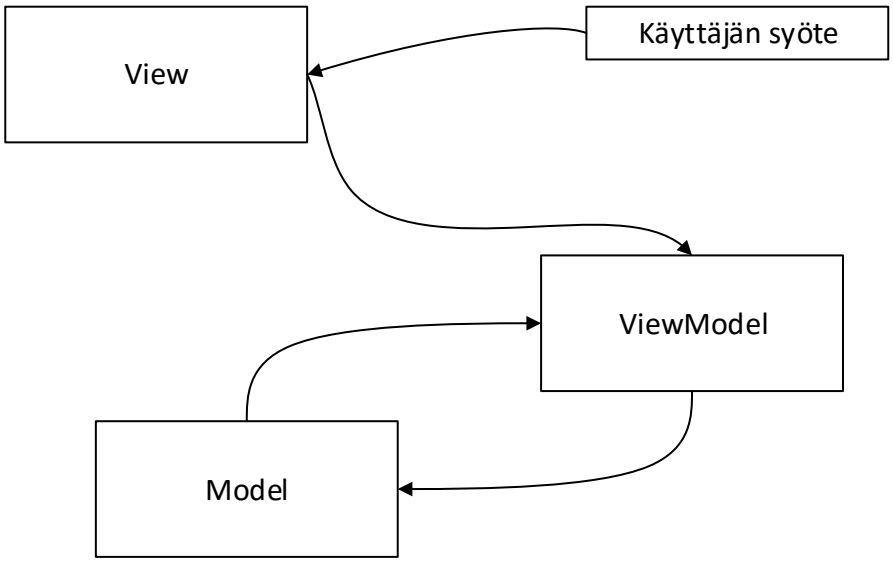
Sovelluksen taustalogiikan kehitys lähti tietomallin määrittelystä, jota seurasi tietokantasuunnittelu. Tietokanta pyrittiin suunnittelemaan niin, että se tukee helposti kaikkia kuviteltuja tulevia tarpeita. Tämän johdosta tietomallista tuli puolestaan myös monipuolinen ja samalla hierarkkinen. Sovelluksen kokonaisarkkitehtuurin kannalta olisi mielekästä, että tietomalli muistuttaisi myös sovelluksen päässä tietokantaa, eikä perustavanlaatuisia eroavaisuuksia ole. Tässä jouduttiin myöhemmässä vaiheessa tulemaan hieman taaksepäin ja yksinkertaistamaan erityisesti asiakkaiden ja tuotteiden esitystapaa käyttöliittymässä näiden aiheuttaman hitauden vuoksi.

Sovellus näyttää listausnäkymässä paljon koontitietoa, jonka täytyy olla valmiiksi laskettua ja helposti saatavilla, jotta käyttäjä listaa nopeasti rullatessaan ei havaitse minkäänlaista nykimistä. Suunnitellun tietokannan rakenne ei palvele tätä käyttötarkoitusta, joten käyttöliittymällä näytettäviä malliluokkia yksinkertaistettiin ja muutettiin tiedot tulemaan kahdessa erässä. Listattaessa esimerkiksi tuotteita, sovellus lataa ainoastaan ne tiedot tuotteista, jotka näytetään listausnäkymässä. Kun käyttäjä avaa tuotteen, ladataan samaan olioon loput tuotteen tiedot, jotka ovat muistissa sovelluksen loppusuoritusajan.

4.3.2 Käyttöliittymäarkkitehtuuri

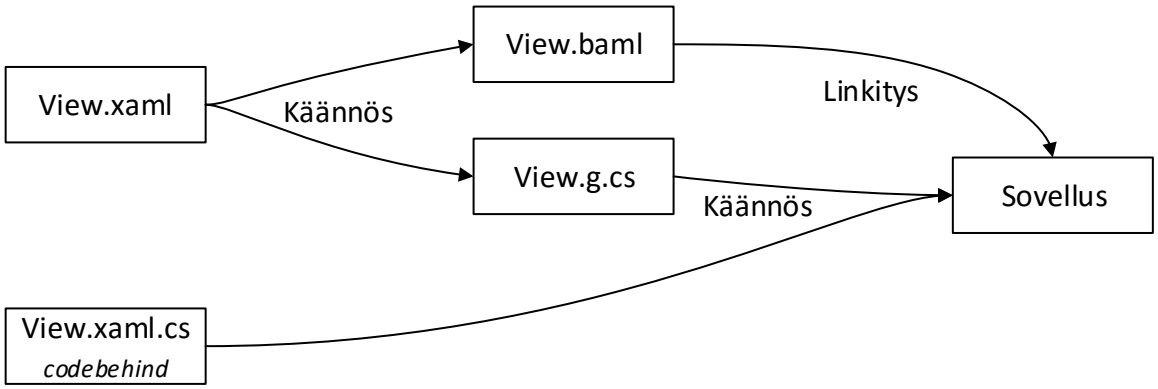
XAML-pohjaiset uudet näkymänkuvauskielet toivat uusien ominaisuuksiensa puolesta myös uusia tarpeita käyttöliittymäarkkitehtuurille. Yksi XAML:in keskeisimmistä piirteistä on löyhä sidonta, jonka ansiosta näkymät voidaan toteuttaa täysin irrallaan sovelluskoodista. Käyttöliittymäelementit voidaan sitoa taustalta tulevaan dataan, ilman että taustan tarvitsee tietää käyttöliittymästä mitään. Tämä mahdollistaa molempien kehittämisen rinnakkain ja näkymien testaamisen testiaineistolla. *MVP*- ja *MVC*-arkkitehtuurit eivät kuitenkaan sellaisenaan sovellu kunnolla tähän tarkoitukseen, sillä niissä kontrollerit näkevät näkymään. [37]

XAML-pohjaisten näkymänkuvauskielten tarpeisiin on MVC-arkkitehtuurista johdettu *MVVM*-arkkitehtuuri, jossa kontrollerin tilalla on näkymämalli, *ViewModel*. Näkymämallilta ei ole suoraa yhteyttä näkymään, vaan näkymä rekisteröi omat elementtinsä näyttämään näkymämallin tarjoamaa dataa ja sitoo omat toimintonsa näkymämallien tarjoamiin komentoihin. Näkymämalli puolestaan keskustelee mallien kanssa, kuten kontrollerit MVC-arkkitehtuurissa ja pääsevät sitä kautta tietomalleihin käsiksi. MVVM-arkkitehtuuri on esitelty kuvassa 10. [37]



Kuva 10: MVVM-arkkitehtuuri [38]

C#-kielellä kirjoitetuissa Windows 8 -sovelluksissa näkymäluokat koostuvat aina kahdesta tiedostosta: XAML-kielisestä näkymäkuvauksesta sekä C#-kielellä kirjoitetusta taustakoodista, *codebehind*:sta. Molemmat esitellään osittaisluokkina, jotka kääntäjä lopulta yhdistää samaksi luokaksi kuvan 11 mukaisesti. Taustakoodiin kirjoitetaan tyypillisesti tapahtumakuuntelijoita tai sieltä voidaan kutsua näkymää elementtien nimillä. MVVM-arkkitehtuuria seurattaessa näin ei tulisi kuitenkaan tehdä, vaan taustakoodi pitäisi jättää tyhjäksi. Näkymälle annetaan konteksti, joka tässä tapauksessa on näkymämalli, jonka kautta näkymä saa kaiken tietonsa. [39]



Kuva 11: XAML ja C# koodin kääntäminen

Windows 8 -sovellusten tapauksessa tämä ei valitettavasti kuitenkaan ole aina kunnolla toteutettavissa, joten myös taustakoodiin joudutaan kirjoittamaan jonkin verran sovelluslogiikkaa joissain tapauksissa. Komentoja, joilla näkymä pystyy viestimään näkymämallien suuntaan, on todella vähän ja tämän vuoksi monesti joudutaan käyttämään tapahtumakuuntelijoita, jotta pystytään suorittamaan ohjelmakoodia tapahtumien johdosta. Eatech Portablessa tähän on käytetty erityistä `EventToCommand`-luokkaa, joka helpottaa tapahtumien muuntamisessa komennoiksi joissain tapauksissa.

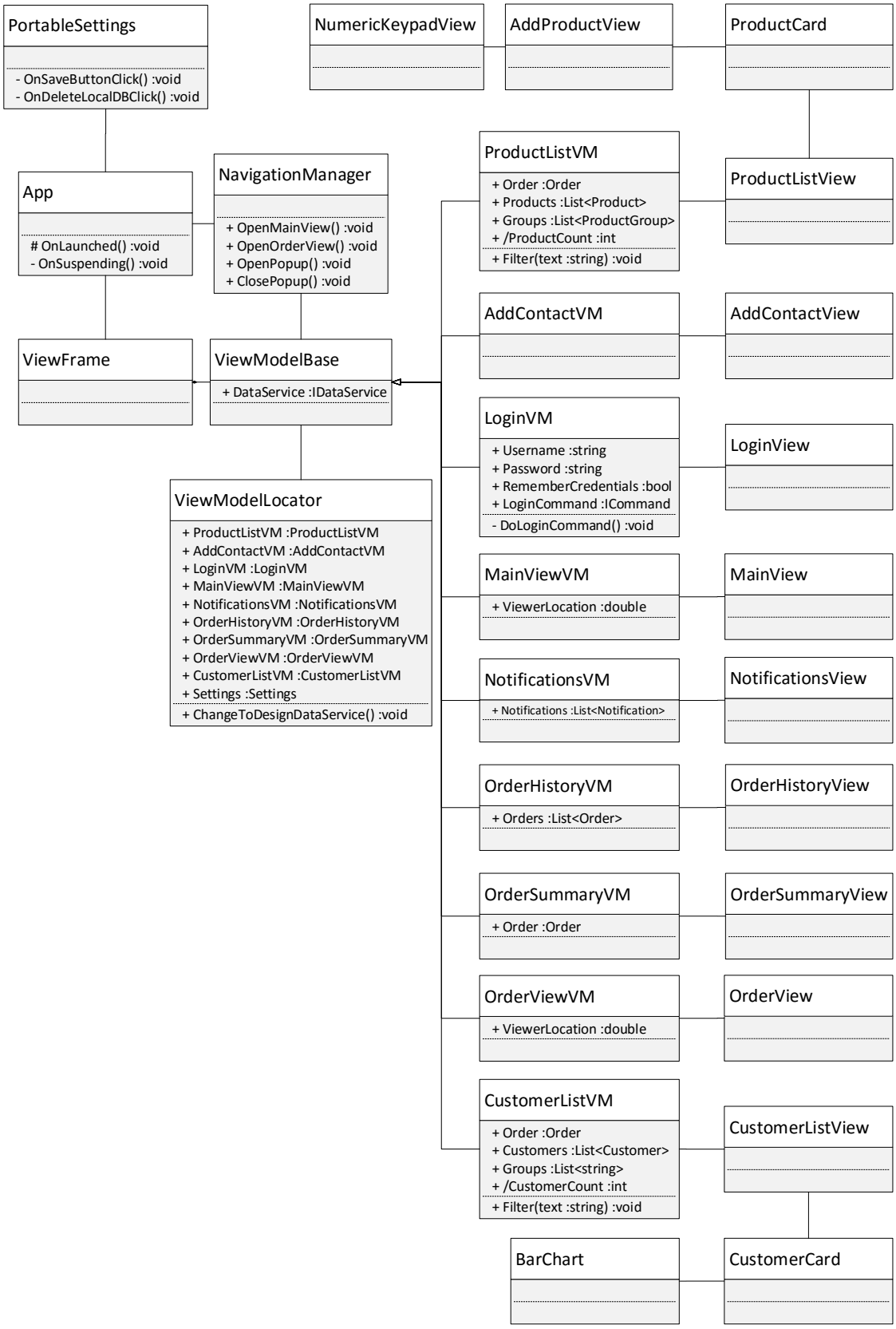
`IDataService` määrittelee rajapinnan DA-kerroksen palveluille, joiden kautta sovellus on yhteydessä tietokantaan ja asiakkaan Eatech Portable -palvelimeen. Demo-tunnuksella kirjauduttaessa käytetään `DesignDataServiceä`, joka tarjoilee käyttöliittymälle `SampleDataSourcen` sisältämää mallidataa. Rajapinta tarjoaa pääsyn dataan *repository-suunnittelumallilla* toteutettujen tietomallien kautta. Rajapinnasta löytyy kokoelmat asiakkaille, tuotteille, tilauksille ja tiedotteille.

DA-kerroksen muista luokista `IWebDataServicen` toteutukset tarjoavat yhteydet asiakkaan palvelimelle ja `IDbDataService` paikalliseen tietokantaan. Molempia näistä käytetään `IDataServicen` kautta, joten nämä ovat vain *TabletClientDA*-projektin sisäiseen käyttöön.

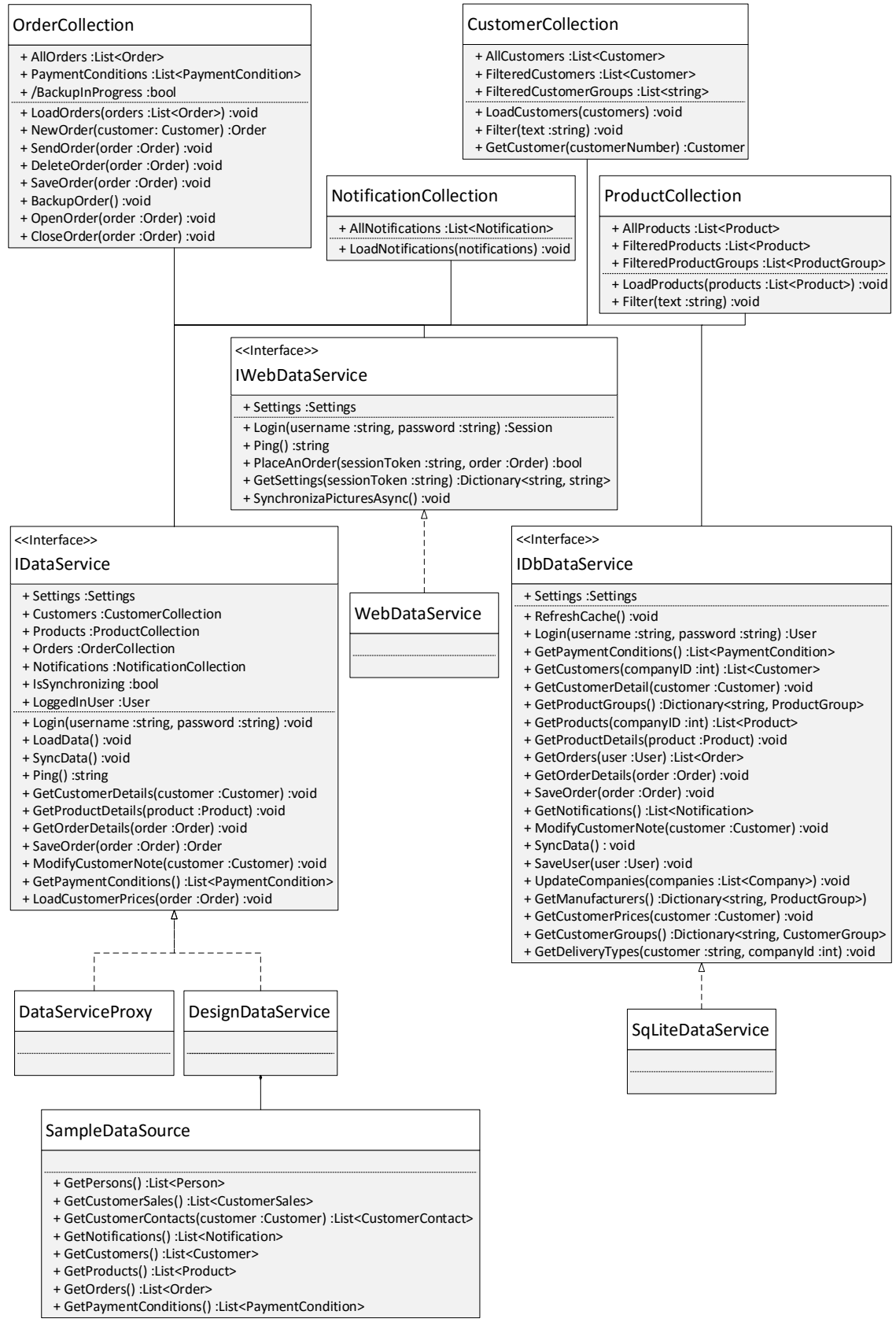
4.3.3 Käyttöliittymän osat

Visual Studiossa sovellus koostuu projekteista, joista yksi on tyypillisesti ajettava sovellus ja loput luokkakirjastoja tai muita sovellukseen liittyviä apuprojekteja. Eatech Portablessa *TabletClient* on varsinainen sovellus, joka referoi kahta luokkakirjastoa. Lisäksi löytyy tietokantaprojekti, jonka avulla pystyy luomaan tietokannan. Kaikki Eatech Portablen projektit ovat:

- **EatechTabletDB** – Tietokanta-projekti. Visual Studio 2010:n mukana julkaistu Database Project helpottaa tietokanta-skriptien hallintaa ja käyttämistä [40]. Eatech Portablen tietokannan luomiseen tarvittavat skriptit on tallennettu taulukohtaisesti tähän projektiin alustusdatan ohella. Muokkauksia tehdessä Visual Studio osaa automaattisesti tehdä vaaditut muunnosskriptit, ilman että tietokannassa olevaa dataa tarvitsee hävittää. Tämä helpottaa merkittävästi kehitystä ja tällä ratkaisulla kehittäjät voivat testata sovellusta omaa lokaalia tietokantaansa vastaan.
- **TabletClient** – Tämä projekti sisältää varsinaisen käyttöliittymän ohjelmakoodin. Projektin luokkakaavio olennaisimmilta osin on esitelty kuvassa 12.
- **TabletClientDA** – Luokkakirjasto, joka sisältää sovelluksen *Data Access* -tason toimintoja. Käytännössä tietokannan käsittely ja palvelimen kanssa kommunikointi tehdään tämän läpi. TabletClientDA:n luokkakaavio on kuvassa 13.
- **TabletInterface** – Sisältää käyttöliittymän ja palvelimen käyttämät tietomallit, joilla varsinainen data siirretään palvelimen ja käyttöliittymän välillä.



Kuva 12: Käyttöliittymän luokkakaavio



Kuva 13: DataAccess-projektin luokkakaavio

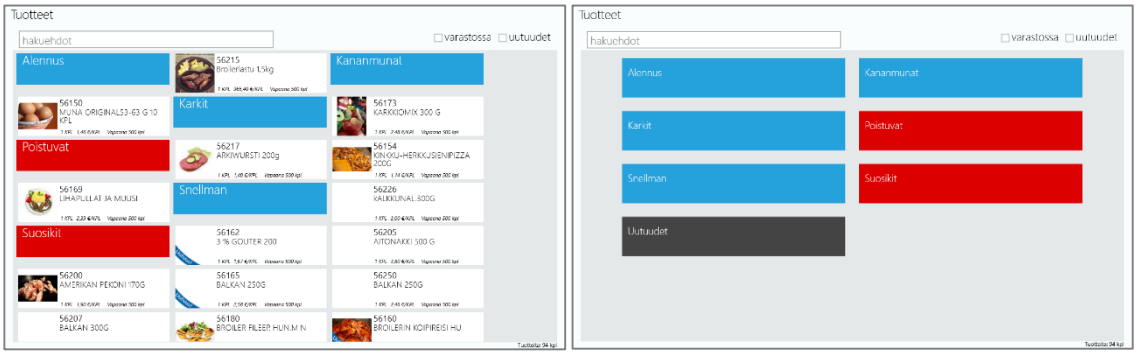
5. TOTEUTUKSEN YKSITYISKOHDAT

Tässä luvussa paneudutaan joihinkin sovelluksen keskeisimmistä yksityiskohdista, jotka ovat ominaisia tälle sovellukselle. Osa näistä on WinRT-alustan tarjoamia valmiita komponentteja ja osa on itse kehitetty sovelluksen tarpeisiin.

5.1 Semanttinen näkymä

Semanttinen näkymä on eräs WinRT:n mukanaan tuomista uusista komponenteista, joka *panoraamanäkymän* ja laatikoiden ohella tuo sovellukseen Windows 8 -sovelluksille tyypillisen käyttökokemuksen. Sen avulla dataan pystyy pureutumaan tarkemmin suurentamalla haluamaansa kohtaa. Tableteilla tämä onnistuu kätevästi nipistys-suurennus-eleen avulla. [41]

Semanttinen näkymä sisältää kaksi kerrosta: `ZoomOut` ja `ZoomIn` -tasot. Eatech Portablessa kontrollia käytetään kahdessa paikassa, asiakkaiden ja tuotteiden listaamisessa. Asiakaslistauksen tapauksessa ylempi taso sisältää listan kirjaimista, joihin suurentaessa tai joita klikattaessa näkymä hyppää aakkosissa kyseisellä kirjaimella alkavien asiakkaiden kohdalle. Tuotelistauksen tapauksessa ylempi kerros sisältää tuoteryhmät. Kuvassa 14 oikealla puolella näkyy tuoteryhmät värikoodattuina ja vasemmalla puolella itse tuotteet ryhmiteltyinä. [41]

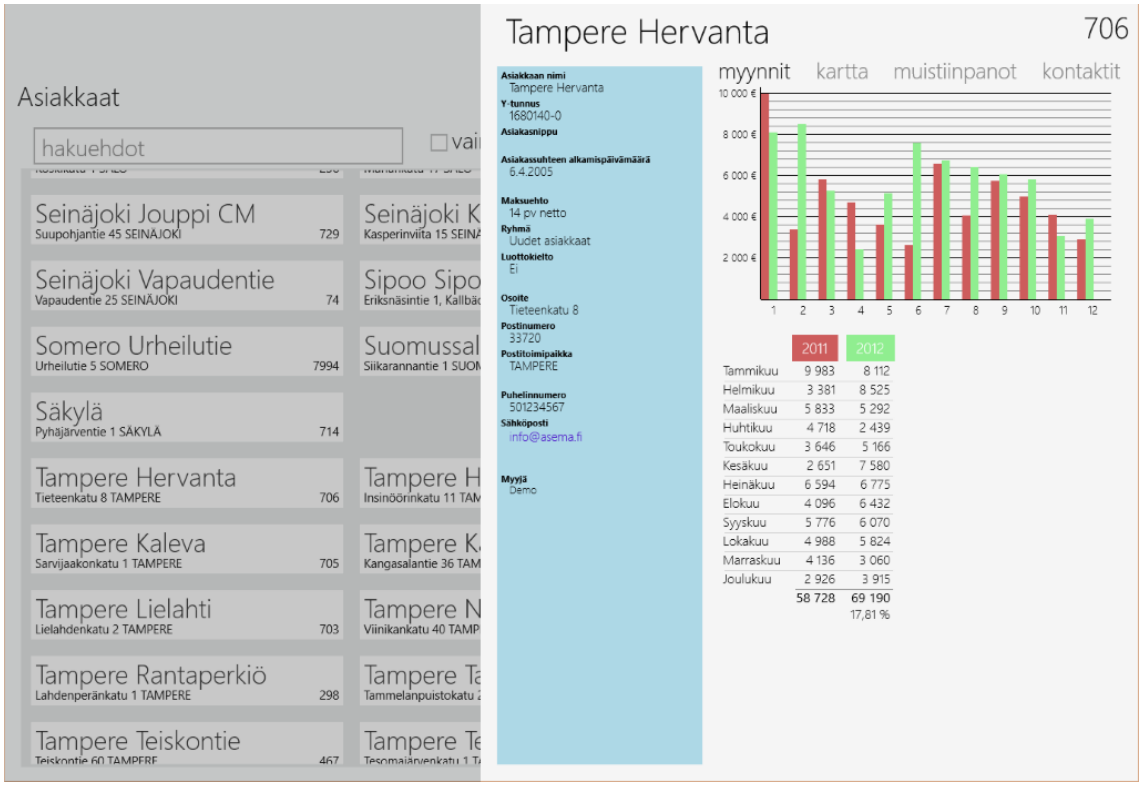


Kuva 14: Semanttinen näkymä

Lisäsimme tuoteryhmiin värikoodauksen, sillä nopeasti selattaessa oikean ryhmän löytää paljon nopeammin värin perusteella, kun sen on oppinut muistamaan.

5.2 Sivupalkki

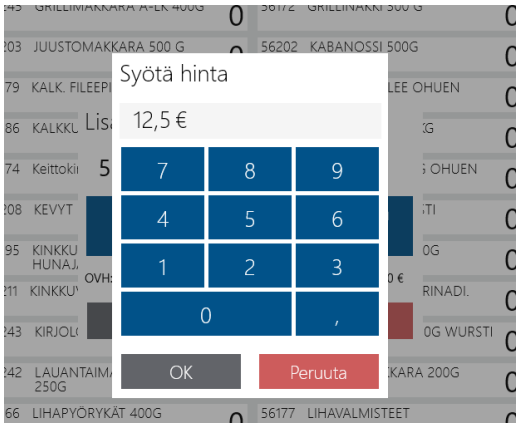
Lisätietojen näyttämiseen esimerkiksi asiakkaista tai tuotteista sovelluksessa käytetään WinRT:n tarjoamia `Flyout`-elementtejä. Nämä voidaan asettaa ponnahtamaan vaikkapa ruudun oikeasta reunasta kuten Eatech Portablessa on tehty kuvan 15 osoittamalla tavalla. Tämän ansiosta käyttäjä ei poistu näkymästä ja käyttökokemus ei katkea. Takaisin siirtyminen tapahtuu intuitiivisesti ohi klikkaamalla ja käyttäjä on heti siinä, mihin jäi. Tämä saa sovelluksen tuntumaan yksinkertaisemmalta, kun ei ole lukuisia eri näkymiä ja animaatioiden avulla tätä saadaan hieman visualisoitua. Näkymä ei nopeasti muutu toisenlaiseksi, vaan esimerkiksi asiakastietokortti tai tuotetietokortti liukuu oikealta ruudulle samalla kun ympäristö himmenee.



Kuva 15: Asiakastietokortti

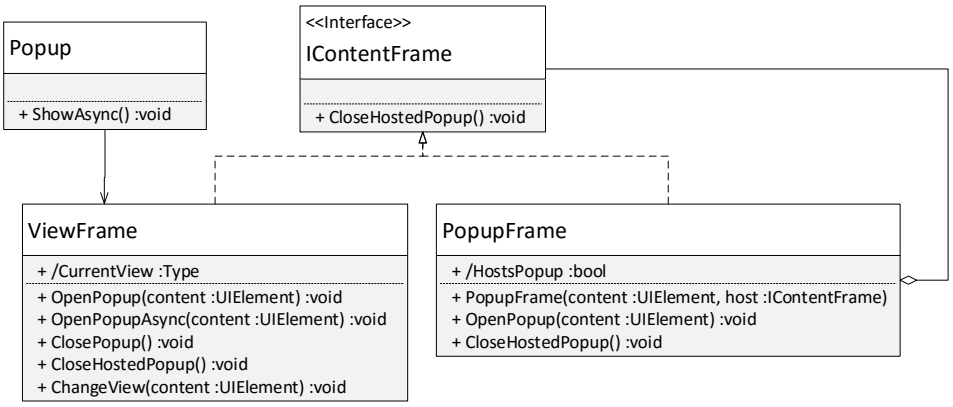
5.3 Dialogit

WinRT:n tarjoamien `Flyoutien` lisäksi sovelluksessa on tarpeita myös popup-tyylisille ikkunoille. Kuvassa 16 näkyy kaksi päällekkäistä popup-ikkunaa, alempana tuotteen lisäysikkuna, jonka päällä on näppäimistö hinnan muuttamista varten.



Kuva 16: Päällekkäiset dialogit

Tällaisia käyttötarkoituksia varten rakensimme sovellukseen tuen modaaleille, päällekkäisille ikkunoille, joiden käyttö onnistuu helposti asynkronisilla kutsuilla ja joiden näkymät voi määritellä täysin vapaasti. Kuvassa 17 näkyy popup-toteutuksen keskeisimmät luokat ja niiden väliset suhteet.



Kuva 17: Dialogin luokkakaavio

Kyseessä on perinteinen rekursiokooste-suunnittelumalli, joka mahdollistaa sisäkkäisten ikkunoiden luomisen. Käyttöliittymä on rakennettu siten, että kaikki mitä ruudulla näkyy, on `ViewFramen` sisällä. `ViewFrame` sisältää kaksi päällekkäistä `Grid`-elementtiä. Alemmassa on varsinainen ruudulla näytettävä näkymä. Tämän vaihtaminen tapahtuu `ViewFramen ChangeView()`-metodilla. Päällimmäiseen `Gridiin` voidaan tuoda näkyville popup, joka sisältää vastaavan kahden päällekkäisen `Gridin` rakenteen.

Mikäli kehittäjä haluaisi luoda esimerkiksi kuvan 16 kaltaisen näppäimistön, loisi hän ensin `UserControl`-elementin, joka sisältäisi näppäimistön ulkoasun sekä toteutuksen ja periyttäisi sen `Popup`-luokasta. Tällöin luodulle elementille voi kutsua `ShowAsync()`-metodia, jonka johdosta kyseinen kontrolli ponnahtaisi ruudulle päällimmäiseksi. Mikäli popup ei käytä koko tilaa, himmennetään sen ympärillä oleva alue ja sen klikkaaminen saa ikkunan sulkeutumaan.

Mikäli modaalilta popup-ikkunalta halutaan paluuarvo, kuten esimerkiksi virtuaalinäppäimistöltä sillä kirjoitettu syöte, voidaan niille tehdä kentät suoraan popup-ikkunan kuvaavaan `UserControlliin`. Koska kutsujalla on kyseinen olio ikkunan sulkeutumisen jälkeen, voi arvot käydä lukemassa sieltä. Asynkroninen toteutus mahdollistaa sen, että suoritus jatkuu `ShowAsync()`-metodin jälkeiselle riville vasta, kun ikkuna on sulkeutunut.

5.4 Lokalisointi

Windows 8 -sovelluksissa lokalisointi tapahtuu WPF-sovelluksista totutulla tavalla resurssitiedostoja käyttämällä, mutta suunta on käänteinen. WinRT:ssä ei enää näkymässä määritellä paikkaa, josta lokalisoidut arvot haetaan, vaan ne pusketaan käyttöliittymäkontrolleihin lokalisointitiedostoista. Tällöin lokalisointeja tehdessä pystytään vaikuttamaan moniin muihinkin kontrollin propertyihin, kuin pelkkään näytettävään tekstiin. Tietyissä tapauksissa on ehkä tarpeen pienentää fonttia, asettaa teksti rivittymään, tai tehdä jotain muita muutoksia oletusarvoihin. [42]

XAML-koodissa kontrolleille annetaan yksilöivä `x:Uid`-tunniste, jonka avulla resurssitiedostosta etsitään arvoja nimellä: `[uid].[property]`. Lokalisointitiedostot tallennetaan kielikoodin mukaisiin kansioihin, jolloin WinRT ottaa ne automaattisesti käyttöön kyseisen kielen ollessa kyseessä. WinRT:n omat kontrollit osaavat myös muuntautua automaattiisesti valitun kulttuurin perusteella, joten esimerkiksi päivämäärät näkyvät automaattisesti oikeassa formaatissa. [42]

Käyttöliittymän tekstien lokalisoinnin lisäksi Eatech Portable sisältää myös paljon esitettävää tietoa, joka tulee pystyä myös lokalisoimaan, esimerkiksi tuotteiden nimet tai niiden tuotetiedot. Päädyimme olemaan tekemättä muutoksia sovelluksen tietomalliin ja siten pitämään tietokannan rakenteen yksinkertaisempana sillä, että emme tuoneet kannan rakenteeseen suoraa tukea lokalisoinneille. Sen sijaan lokalisointi tapahtuu tallentamalla pelkän tekstin sijasta haluttuun kenttään JSON-muodossa halutut käännökset kielikoodeineen. Tällöin käyttöliittymä valitsee valitun kulttuurin perusteella oikean tekstin ja palvelimen päässä riittää, että integraatiopalvelu osaa muodostaa oikeanlaisia JSON-tietueita tietokantaan.

5.5 Virhetilanteiden hallinta

Koska sovelluksen läpi voi mennä parhaimmillaan suurin osa asiakasyrityksen tilauksista, on selvää, että kyseessä on liiketoiminnan kannalta varsin kriittinen sovellus. Tämä näkyy erityisesti sovelluksen laadun varmistamisena ennen julkaisua tehtävissä kattavassa testaamisessa, mutta myös odottamattomiin virheisiin täytyy pystyä varautumaan.

Asiakasyritysten myyjät saattavat tehdä asiakkaidensa luona tilauksia, jotka sisältävät satoja rivejä tuotteita ja tilauksen tekemiseen käytetään merkittävästi aikaa. Jotta mahdollinen sovelluksen kaatuminen, akun loppuminen tai jokin muu odottamaton tilanne ei tuhoa tehtyä tilausta, tekee sovellus minuutin välein tilauksesta automaattisesti varmuuskopion paikaliseen kantaan. Mikäli sovellus syystä tai toisesta sammuisi, pystyy tilauksen tekemistä jatkamaan siitä mihin jäätiin.

Mikäli tilauksen lähetys epäonnistuu, tallentuu se myös lokaalisti ja tilaksi jää ”virhe lähetyksessä”. Käyttäjä voi myöhemmin yrittää lähetystä uudelleen, mikäli kyseessä olisi ollut tietoliikenne- tai jokin muu ohimenevä ongelma. Automaattisesti uudelleenlähetystä ei kuitenkaan tehdä, jottei tilausta tehtäisi vahingossa kahta kertaa myyjän tietämättä. Sovelluksessa on pidetty huolta, että tilaus lähetetään palvelimelle vain kerran ja että mikäli käyttäjä näkee tilauksen lähetyksen onnistuneen, niin silloin sen voi myös luottaa päässeen perille. Tämä on tärkeää, koska myyjällä on oltava tieto mitkä tilaukset ovat varmasti päässeet perille asti, ilman että asiasta on epäselvyyttä.

WinRT tarjoaa sovellustason tapahtuman (engl. *event*) nimeltä `UnhandledException`, joka laukeaa aina kun sovelluksesta on vuotamassa poikkeus käyttöjärjestelmälle, joka puolestaan tappaa sovelluksen tämän johdosta. Tätä kuuntelemalla saadaan suoritettua koodia juuri ennen sovelluksen kaatumista. Tällaisen tapauksessa sovellus kerääkin sovelluksen kaatavasta poikkeuksesta sen tyypin, virheviestin, kutsupinon sekä muutamia muita virheen selvittämisessä hyödyllisiä tietoja ja lähettää ne Web Service -yhteyden yli palvelimelle. Tämän ansiosta pääsemme ennakoivasti kiinni sovelluksen kaatumisiin ennen kuin sovelluksen käyttäjät niistä välttämättä kerkeäisivät raportoimaankaan ja virheiden paikallistaminen on nopeampaa. Yhdistettynä tämä Windows Storen kautta jaeltaviin ja oletuksena automaattisesti asentuviin päivityksiin, saadaan korjaus parhaimmissa tapauksissa tunneissa pihalle virheen muodostumisesta.

Kaikki ongelmat eivät kuitenkaan ratkea pelkästä sovelluksen kaatavasta poikkeuksesta ja sen tiedoista, vaan myös logituksella on merkittävä rooli virheiden korjaamisessa. Sovellus pitää logia tapahtumistaan paikalliseen SqLite kantaan. Tästäkin olisi haluttaessa lähetettävissä esimerkiksi osa palvelimelle kaatumisen yhteydessä, mutta toistaiseksi tällaista toiminnallisuutta ei ole tehty. Mikäli laitteen logeihin halutaan päästä kiinni, pitää käyttäjää pyytää lähettämään kyseinen tietokantatiedosto tai se pitää noutaa itse etäyhteyden avulla.

5.6 Asiakaskohtaisten muutosten hallinta

Heti alkuun oli selvää, että yksi keskeisimmistä haasteista, joka tulisi ratkaista Windows Store –versiossa, on sovelluksen räätälöitävyys asiakkaiden tarpeiden mukaan.

Windows Store -jakelun etuihin kuuluu kiistämättä sen helppous. Päivityksiä ei tarvitse manuaalisesti asentaa myyjien koneille, jonka tekeminen varsinkin käsin on erittäin työlästä ja aikataulutuksen puolesta haasteellista. Toisaalta asiakkaat haluavat monesti, että sovellusta muokataan heidän tarpeisiinsa ja näiden räätälöintien tekeminen kaikille asiakkaille jaeltavaan yhteiseen versioon, niin että muutokset eivät vaikuta muihin, tekee niiden tekemisestä monimutkaisempaa ja siten myös kalliimpaa.

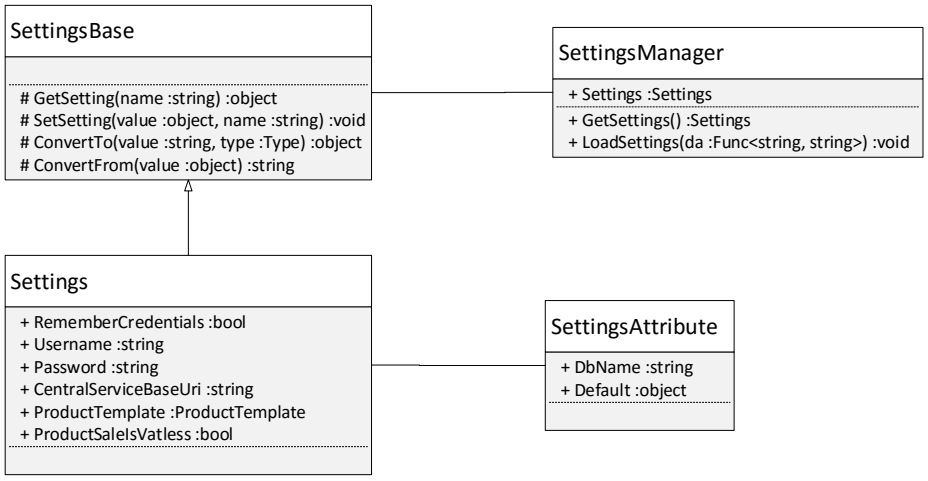
Mikäli asiakas toivoo merkittäviä muutoksia sovelluksen toimintalogiikkaan ja sideloading-vaatimukset täyttyvät asiakkaan IT:n puolesta, on sovelluksen haaroittaminen master-haarasta ja Windows Storen ohi jakaminen varteenotettava vaihtoehto.

5.6.1 Asetusten lataaminen

Sovelluksessa on karkeasti kolmenlaisia tarpeita asetuksille. Näitä ovat:

- paikalliset, laitteella säilytettävät asetukset tai tiedot,
- sovelluksen kovakoodatut vakiot sekä
- asiakkaan palvelimelta ladattava räätälöintidata, sisältäen graafisen ulkoasun ja toiminnallisuusmuutokset.

Näihin tarpeisiin kehitimme kuvassa 18 näkyvät `Settings`- sekä `SettingsManager`-luokat riippuvuuksineen. Vastaava toiminnallisuus onnistuisi yhdellä luokallakin suhteellisen helposti, mutta päätimme sovelluksen ylläpidettävyyden vuoksi tehdä asetusten hallinnasta kehittäjälle mahdollisimman helppoa ja sen vuoksi kätkeä kaiken näihin liittyvän toiminnallisuuden taustalle. Nyt kehittäjän tarvitessa mitä tahansa näistä kolmesta edellä mainitusta asetustyypistä, riittää että hän esittelee sen `Settings`-luokassa.



Kuva 18: Asetusten käsittelyn luokkakaavio

`Settings`-luokka säilötään DA-tasolle `IDataService`-rajapinnan taakse, ja se on siten saatavilla `ViewModelLocatorin` avulla. Esimerkiksi `ViewModeleille` tämä tulee *Dependency Injectionilla* suoraan luokan rakentajalle.

Vaikka kyseisellä tavalla asetuksiin pääseekin kiinni lähes kaikkialta sovelluksesta yhtä helposti, päädyimme määrittelemään vakiot staattisiksi, jolloin niiden käyttö onnistuu suoraan luokan kautta. Tämä on tarpeen esimerkiksi aivan sovelluksen käynnistyksen yhteydessä, jolloin `Settings`-toteutus ei muuten olisi vielä käytettävissä.

Sovelluksessa on useita rutiineja, joissa saatetaan haluta tallentaa paikallisesti tietoja. Yksi esimerkki tällaisesta on käyttäjän kirjautumistietojen muistaminen. Tiedot tallennetaan ja luetaan sovelluksen `LocalCache`-kansioon tallentamasta .xml-tiedostosta. Asetus määritellään `Settings`-luokkaan tavallisena C#-propertynä, jonka `get`- ja `set`-toteutuksiin kirjoitettavat geneeriset `GetSetting()` ja `SetSetting()` -metodit vastaavat tiedon lataamisesta, tallentamisesta sekä tarvittavista muunnoksista.

Asiakaskohtaisten räätälöintien tekemistä varten tarvitaan myös kolmas menetelmä, joka myös näkyy C#-propertynä ulospäin, mutta paikallisesti tallennettavista asetuksista poiketen näihin tehdyt muutokset eivät tallennu. `SettingsManager` pyrkii alustuksensa yhteydessä lataamaan palvelimelta arvot kaikkiin attribuutilla merkittyihin propertyihin. Mikäli palvelimen tietokannassa ei kyseistä asetusta ole, käytetään attribuutissa määriteltyä oletusarvoa. Ladatut arvot tallennetaan myös paikalliseen SqLite-tietokantaan, jotta nämä olisivat käytettävissä myös siinä tapauksessa, että sovellus ei saa käynnistyessään yhteyttä palvelimeen.

Listauksessa 2 esitellään kaikki kolme tapaa käyttää `Settings`-toteutusta. Tämä on samalla myös täysin toimiva luokka, joka mahdollistaa käyttäjänimen tallentamisen ja lukemisen (rivi 4), sisältää staattisen tietokantatiedoston nimen, jota tarvitaan sovelluksen käynnistymisen yhteydessä (rivi 7), sekä palvelimelta ladattavat räätälöintitiedot ALV:n käytöstä (rivit 10-11) ja joitain tyylejä, joilla voi muokata sovelluksen värimaailmaa (rivit 14-18). Mikäli palvelimen tietokantaan ei ole tallennettu esimerkiksi `Background`- tai `AccentColor`-asetuksia, käytetään oletusarvoja.

```
 1:  public class Settings : SettingsBase
 2:  {
 3:    //Paikalliselle laitteelle tallennettava arvo
 4:    public string Username {get{return GetSetting();}set{SetSetting(value);}}
 5:
 6:    //Vakioiden määrittely
 7:    public static readonly String DatabaseName = "PortableLite.db";
 8:
 9:    //Asiakkaan palvelimelta ladattavat kustomoidut arvot
10:    [Setting(Default = true)]
11:    public Boolean ProductSaleIsVatless { get; set; }
12:
13:    //Muun muassa ulkoinen ilme pystytään lataamaan asiakkaan palvelimelta
14:    [Setting(Default = "#FFFAFDFD")]
15:    public String Background { get; set; }
16:
17:    [Setting(Default = "#015289")]
18:    public String AccentColor { get; set; }
19:  }
```

Listaus 2: Asetusten käyttäminen koodissa

5.6.2 Mallit ja muuntimet

Räätälöintiin tarvittavien asetusten helppo määrittely ja käyttäminen koodissa selkeyttää sovelluksen rakennetta, mutta niistä on sellaisenaan apua ainoastaan ehdollisissa koodirakenteissa, joiden pohjalta sovelluslogiikka toimii hieman eri tavalla. Yksi esimerkki voisi olla vaikkapa arvonlisäveron käyttäminen, joka vaatii yhden ylimääräisen ehtorakenteen muutamiin näkymämalleihin ja malleihin. Monimutkaisiin tapauksiin tarvitaan tehokkaampia menetelmiä.

Mallien (engl. *template*) avulla pystytään tekemään käyttöliittymäkomponenteille erilaisia esitystapoja. Sovelluksessa tuotelistauksessa näytetään oletustoteutuksessa kuva, tuotteen nimi, varastosaldo ja muutamia muita tietoja. Mikäli asiakas haluaisi muokata tuotteen näyttämistapaa listalla, voitaisiin sille luoda uusi malli. Tällöin sovellus sisältäisi kaksi esitystapaa tuotteiden listaamiseen. Lisäksi tarvitaan XAML:n tarjoama `TemplateSelector`, jolle voidaan antaa parametriksi alakohdassa 5.6.1 esitellyn `Settings`-toteutuksen avulla ladattu tieto halutusta tuotemallista.

Mikäli esitystapoja on useita, lisää sekin sovelluksen monimutkaisuutta, sillä jotkin muutokset sovellukseen täytyy tehdä näihin kaikkiin. Jos näkymä pysyy lähes samana ja muutokset liittyvät vain muutamaan kohtaan, voi kyseeseen tulla muuntimien käyttäminen.

Palvelimelta ladatut asetukset harvoin kelpaavat sellaisenaan moneenkaan käyttöliittymän räätälöimistarkoitukseen, vaan sille täytyy tehdä kohteen perusteella muunnoksia. Ladattaessa asetuksista enumeraatio, joka ilmaisee halutun vaihtoehdon esittää esimerkiksi tuotelistauksen tuote-laatikon mitat, voitaisiin näiden arvo laittaa tulemaan suoraan kyseisestä asetuksesta XAML:n Data Binding:in avulla, mutta määritellä sille lisäksi muunnin, joka palauttaa mitat saadun enumeraation arvon perusteella.

6. ARVIOINTI

Tässä luvussa arvioidaan projektin onnistumista ja käydään läpi tekijöitä, jotka ovat osaltaan vaikuttaneet projektin onnistumiseen ja toisaalta mitkä asiat ovat luoneet haasteita. Lisäksi tehdään tarkempi katsaus WinRT-sovelluskehykseen.

6.1 Toteutuksen onnistuminen

Projektin alkuvaiheessa tunnistettiin joitakin haasteita, joita sovelluskehityksessä on odotettavissa. Kaikki nämä kohdassa 2.3 luetellut haasteet saatiin ratkaistua, mutta osa niistä realisoitui, mikä näkyi työmääräarvioiden ylittymisenä. Sovellus valmistui kuitenkin aikataulussa ja se saatiin asiakkaalle käyttöön toivotusti.

WinRT-ohjelmistorajapinnan avulla on helppo toteuttaa tableteille pienellä vaivalla näyttäviä kosketuskäyttöliittymiä, minkä ansiosta useiden käyttöliittymähahmotelmien kokeileminen oli vaivatonta. Sovelluksen käyttöliittymästä saatiin uusien Windows 8 -kontrollien avulla yksinkertainen ja nopea.

Vaikka sovelluksen käyttöliittymän rakenne mahdollisti nopean käyttämisen, aiheutti suuri tietomäärä merkittäviä hidasteita. Käynnistämiseen meni ensimmäisillä versioilla minuutteja, sovelluksen tilojen välillä siirtyminen kesti kauan ja asiakas- ja tuotelistauksien nopea selaaminen ei onnistunut. Muun muassa suorituksen rinnakkaistamisen, tietomallien yksinkertaistamisen ja muiden rakenteellisten muutosten ansiosta nämä ongelmat saatiin ratkaistua. Käynnistysnopeudet saatiin laskettua sekunteihin, siirtymät sovelluksen sisällä tapahtuvat välittömästi ja tuhansia rivejä sisältäviä listoja pystyy rullaamaan niin nopeasti kuin haluaa, ilman että näkymä alkaa nykiä. Myös synkronointi pystyttiin toteuttamaan Microsoft Sync Frameworkin avulla niin tehokkaaksi, että päivittäin tehtynä siihen meni parhaimmillaan vain sekunteja.

Windows Runtime -alusta oli kuitenkin projektin alkaessa niin keskeneräinen, että sovelluksen arkkitehtuurin kanssa jouduttiin tekemään monia kompromisseja, jotka sovelluskehyksen muistivuotojen ja muiden bugien kanssa näkyivät työmääräarvioiden ylittymisenä. Myös käyttöliittymäsuunnitteluun käytettiin enemmän aikaa, kuin oltiin arvioitu. Tässä vaiheessa oli kuitenkin tiedossa, että sovelluksesta halutaan myös oma tuote, joten tämä nähtiin kannattavana investointina.

Projektin voi tulkita onnistuneeksi, mikäli mittarina käytetään asiakastyytyväisyyttä ja sovellukselle asetettujen vaatimusten täyttymistä. Asiakas, jolle sovellusta aloitettiin alun perin tekemään, on ollut erittäin tyytyväinen lopputulokseen ja sovellusta on sen jälkeen jatkokehitetty asiakkaan kanssa useampaan otteeseen. Sovellus voitti Microsoftin vuonna

2014 järjestämän *App Awards* -sovelluskilpailun business-sarjan. Myös kaikki sovellukselle asetetut vaatimukset saatiin täytettyä ja kohdassa 2.3 listattuihin haasteisiin vastattua. Sovelluksen navigaatio on saanut kiitosta, erityisesti tuotteiden selailu ja ostoskorin käyttö on nopeaa. Myös synkronointi ja tiedonsiirto tapahtuvat todella nopeasti.

Käyttöliittymän toteutuksen tekniseen onnistumiseen vaikuttaa erityisesti siihen investoitu lisäpanos. Tämän lisäksi lopputuloksen kannalta oli hyvä, että jonkinlainen järjestelmä oli ollut olemassa aikaisemmin, jotta asiakas tiesi tarpeensa hieman tarkemmin. Koska alusta ja laitteen tyyppikin vaihtuivat, saimme toteutukselle melko vapaat kädet. Tämä mahdollisti aivan uusien ratkaisuvaihtoehtojen etsimisen, joten pystyimme panostamaan erityisesti käytettävyyteen.

Asiakasprojektin valmistumisen jälkeen sovellusta alettiin kehittää omana tuotteena ja Windows 8.1 julkaisun jälkeen sovellukseen pystyttiin tekemään suurempi arkkitehtuurimuutos, jolloin aikaisemmin tehdyt tilapäiset ratkaisut korvattiin paremmilla. Tämän ansiosta asennus yksinkertaistui, sovellus pystyttiin julkaisemaan Windows Storessa ja siihen saatiin parempi tuki räätälöinneille, mikä oli edellytyksenä uusien asiakkaiden saamiseksi.

6.2 Referenssit

Sovelluksella on reilun kaksivuotisen ikänsä aikana useita käyttäjiä, joista osalta on julkinen referenssi nähtävillä Eatechin verkkosivuilla.

Prima Pet Premium Oy:n kehityspäällikkö Lari Majamäki korostaa sovelluksen offline-tuen merkitystä ja sovelluksen potentiaalia: *”Sovelluksen offline-mahdollisuus on yksi sen vahvuuksista. Pro-meininkiä on myös se, kun näkee varastosaldot ja ostotilauskannan. Ei tarvitse myydä EI-OO:ta. Uskallan väittää, että myyntisovelluksella voi todella MYYDÄ tuotteita ja palveluita, eikä vain ottaa tilausta asiakkaalta. Samaa olen kuullut meidän myyjiltä. Sovelluksen käyttöliittymä on selkeä ja toimintalogiikka johdonmukainen. Vanhan liiton reppurikin oppii sovelluksen käytön hetkessä.”* [43]

Miraculos Oy:n toimitusjohtaja Jyrki Hakala kertoo yrityksen pystyneen tehostamaan liiketoimintaansa sovelluksen avulla: *”Myyntisovelluksen avulla olemme tehostaneet päivittäistä työtämme. Nyt myyjillä on aina ajantasainen tieto mukanaan ja he voivat esitellä tuotteitamme näyttävästi. Sovelluksen käyttö on äärimmäisen helppoa ja sen avulla tilaukset saadaan nopeasti kirjattua ja toimitettua asiakkaalle. Eatechin avulla saimme tehostettua liiketoimintaamme ja otettua liiketoimintamme tueksi modernin ratkaisun.”* [44]

6.3 WinRT-kehityksen rajoitteet ja haasteet

Sovelluksen toteuttamista hidasti merkittävästi Windows Runtime -alustan nuoruus, monin paikoin jopa keskeneräisyys. Monet aikaisemmista Microsoftin sovelluskehyksistä löytyneet toiminnallisuudet puuttuivat kokonaan ja monet asiat toimivat väärin tai olivat epävakaita. Lisäksi uuden ympäristön asettamat tiukat rajoitteet asettivat suuria haasteita. Isoin näistä oli tietokannan käsittelyyn liittyvä toiminnallisuus, joka pakotti kokonaisarkkitehtuurin aluksi paljon monimutkaisemmaksi, kuin mille olisi ollut tarve.

Puuttuvien kontrollien ja komponenttien lista oli pitkä. WinRT XAML Toolkit paikkasi osin tätä, mutta osa komponenteista jouduttiin toteuttamaan itse. Esimerkiksi välilehtikontrollia ei löytynyt Microsoftin, eikä kenenkään muunkaan toteuttamana. Sama koski popup-ruutuja, muokattavia dialogeja ja esimerkiksi päivämäärävalitsimia. Viimeisin tuli Windows 8.1 -päivityksen myötä saataville.

Vaikka XAML-kuvauskieli on hyvin lähellä WPF:stä ja Silverlightista löytyvää XAML:ia, ei Microsoft ollut aivan kaikkia ominaisuuksia ottanut sieltä. Keskeisimpinä puutteina `StringFormat`, jolla olisi suoraan XAML-koodissa pystynyt muokkaamaan tiedon esitysmuotoa. Nyt vastaava tehtävä suoritetaan erikseen tätä varten toteutetulla muuntimella. Myöskään tukea `MultiBindingille` ei ole, jolla näkymän näyttämä tieto olisi saatu tulemaan useamman arvon yhdistelmän perusteella. Myös tämä on ratkaistavissa erikseen ohjelmoitavilla muuntimilla, mutta nämä heikentävät koodin luettavuutta.

Vaikka WinRT:n tarjoama `ICollectionView`-rajapinta tarjoaa rajapintametodit tiedon järjestämiselle ja suodattamiselle, ei tätä tukea kuitenkaan WinRT:n kyseisen rajapinnan toteutuksista löydy, vaan jouduimme rakentamaan omat tietorakenteet. Myöskään ryhmittely ei onnistunut `CollectionView:n` kautta semanttista näkymää käyttäessä, vaan sekä ryhmille että tuotteille/asiakkaille jouduttiin tekemään erilliset listat ja siirtyminen näiden kahden välillä toteuttamaan ohjelmallisesti. Semanttisen näkymän avulla tämän pitäisi tapahtua automaattisesti, mutta Windows 8:n julkaisuversiossa siinä oli vielä pahoja virheitä, eikä se ollut käytettävissä kaikilla laitteilla.

Myös ympäristön muistivuodot aiheuttivat hankalasti selvitettäviä ja ratkaistavia ongelmia. Muutamassa tapauksessa komponentista oli tehtävä *singleton*-instanssi ja pidettävä se sovelluksen muistissa sovelluksen suorituksen ajan, tai sovelluksen sisällä liikkuminen olisi lopulta johtanut kaatumiseen. Yksi esimerkki on Bing Maps -karttakontrolli, joka ei tuhoutuessaan vapauttanut kaikkea sisäisesti varaamaansa muistia. Lisäksi WinRT-alustan tarjoama virtualisointituki jouduttiin kytkemään muutamassa tapauksessa pois päältä samasta syystä.

7. YHTEENVETO

Liikkuva myyjä kohtaa myyntikäyntiensä yhteydessä useita haasteita, joihin tietoteknisillä ratkaisuilla pystytään tuomaan helpotusta. Tässä diplomityössä käsitelty Eatech Portable Sales pyrkii tarjoamaan liikkuville myyjille työkalun, joka sisältää kaiken tiedon ja toiminnot, joita myyjät tarvitsevat asiakaskäyntinsä yhteydessä. Tehokkaan ja ilmaisuvoimaisen sovelluksen tavoitteena on nopeuttaa asiakkaan luona tehtävää myyntiprosessia sekä lisätä myyntiä ja siten tehostaa liiketoimintaa.

Tässä työssä on pyritty kuvaamaan Eatech Portablen Sales –tablet-sovelluksen suunnittelu- sekä toteutusprosessin. Diplomityön tekeminen alkoi vasta kaksi vuotta sovelluksen julkaisemisen jälkeen kirjoittajan muiden töiden vuoksi. Kirjoitusprosessi painottui lähinnä syksyyn ja eteni vauhdilla, kun aiheeseen pääsi kunnolla syventymään. Työn avulla sovellusta jatkokehittävät kehittäjät pääsevät luultavasti nopeammin sisälle sovelluksen arkkitehtuuriin ja ymmärtävät paremmin sovelluksessa tehtyjä arkkitehtuuriratkaisuita.

Asiakkaalla, jolle sovellus alkujansa tehtiin, oli aikaisemmin käytössä Windows Mobile -sovellus samaan käyttötarkoitukseen. Kyseiset laitteet ja sovellus eivät kuitenkaan vastanneet kunnolla asiakkaan tarpeisiin, jonka johdosta tablet-sovellusta ruvettiin rakentamaan.

Alustaksi sovellukselle valittiin vasta julkaistu Windows 8, koska Microsoftin käyttöjärjestelmät ovat yrityskäytössä käytetyimpiä. Myyjän on mahdollista hankkia hybridilaite, jolloin hän pystyy samalla laitteessa tekemään kaikki työt, joihin on tarvinnut aikaisemmin kannettavaa ja samalla se soveltuu tablet-tilassa asiakkaiden luona tehtävään myyntityöhön.

Windows 8 -alusta aiheutti suuria haasteita uutuutensa ja sen puolesta, että sitä ei oltu vielä suunnattu yrityskäyttöön. Monet WinRT-kehyksen tarjoamat komponentit olivat keskeneräisiä, sisälsivät muistivuotoja ja muita virheitä. Puuttuvia komponentteja jouduttiin itse toteuttamaan ja arkkitehtuuriratkaisua monimutkaisti Windows 8:n rajoitukset.

WinRT mahdollisti kuitenkin hyvin sen, mihin se oli suunniteltu: käytettävien ja visuaalisten sovellusten tekemisen. Sovelluksella saavutettiin kaikki tavoitteet ja asiakkaat ovat olleet tyytyväisiä lopputulokseen. Sovellusta on jatkokehitetty sen julkaisun jälkeen ja arkkitehtuuria on refaktoroitu parempaan suuntaan alustan kehittyessä. Sovelluksen kehitys ja myynti jatkuvat edelleen ja siitä on harkinnassa myös versiot muille alustoille.

LÄHTEET

[1] Global market share held by operating systems Desktop PCs from January 2012 to June 2015. Statista. [viitattu 25.10.2015]. Saatavissa: http://www.statista.com/statistics/218089/global-market-share-of-windows-7/.

[2] Browne, B. Five Lessons from a Year of Tablet UX Research. [viitattu 25.10.2015]. Saatavissa: https://uxmag.com/articles/five-lessons-from-a-year-of-tablet-ux-research.

[3] Lux, A. Yesterday's Tomorrows: The Origins of The Tablet. [viitattu 25.10.2015]. Saatavissa: http://www.computerhistory.org/atchm/yesterdays-tomorrows-the-origins-of-the-tablet/.

[4] Thurrott, P. Tablet PC Redux? [viitattu 25.10.2015]. Saatavissa: http://winsupersite.com/mobile-devices/tablet-pc-redux.

[5] Windows Hardware Certification Requirements. MSDN. [viitattu 5.11.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/windows/hardware/dn423132.

[6] Bajaj, K. From DLNA to Wi-Fi, NFC and Bluetooth electronic devices heading to a wire free world. [viitattu 15.11.2015]. Saatavissa: articles.economictimes.indiatimes.com/2012-01-04/news/30588923.

[7] The Evolution of Near Field Communication (NFC). [viitattu 25.10.2015]. Saatavissa: http://www.techpats.com/evolution-near-field-communication-nfc/.

[8] John M. The WinRT Location API: Where did my location data come from? Intel. [viitattu 25.10.2015]. Saatavissa: https://software.intel.com/en-us/blogs/2012/09/11/the-winrt-location-api-where-did-my-location-data-come-from.

[9] Djuknic, G.M. & Richton, R.E. Geolocation and assisted GPS. Computer 34(2001)2, pp. 123-125.

[10] Processors 101: Understanding ARM and Windows RT. Staples. [viitattu 25.10.2015]. Saatavissa: http://www.staples.com/sbd/cre/tech-services/explore-tips-and-advice/tech-articles/processors-101-understanding-arm-and-windows-rt.html.

[11] Intel Atom Vs Intel Core i7 – Which processor is best for your Tablet PC? [viitattu 15.11.2015]. Saatavissa: http://blog.tabletpc.com.au/2010/07/09/intel-atom-vs-intel-core-i7-which-processor-is-best-for-your-tablet-pc/.

[12] Linenberger, M. The Importance of an Active Digitizer Pen. [viitattu 9.11.2015]. Saatavissa: http://www.michaellinenberger.com/blog/the-importance-of-an-active-digitizer-pen/.

[13] Microsoft Microphone Array Support in Windows. [viitattu 9.11.2015]. Saatavissa: http://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/micarrays.doc.

[14] Ambient-Light Sensing Optimizes Visibility and Battery Life of Portable Displays. [viitattu 9.11.2015]. Saatavissa: https://www.maximintegrated.com/en/app-notes/index.mvp/id/5051.

[15] Mayberry, M. WinRT Revealed. 2012, XVII, 92 p

[16] Kendrick, J. Windows 8 without touch is like a day without sunshine. ZDNet. [viitattu 5.11.2015]. Saatavissa: http://www.zdnet.com/article/windows-8-without-touch-is-like-a-day-without-sunshine/.

[17] Whitney, L. Why did Microsoft kill the name 'Metro'? [viitattu 25.10.2015]. Saatavissa: http://www.cnet.com/news/why-did-microsoft-kill-the-name-metro/.

[18] Microsoft Capabilities. MSDN. [viitattu 25.10.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/windows/apps/br211422.aspx.

[19] Microsoft App lifecycle. MSDN. [viitattu 25.10.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/windows/apps/xaml/hh464925.aspx.

[20] Microsoft How to enable loopback and troubleshoot network isolation. MSDN. [viitattu 25.10.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/windows/apps/Hh780593.aspx.

[21] Microsoft Asynchronous Programming with Async and Await. MSDN. [viitattu 25.10.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/hh191443.aspx.

[22] Asynchronous IO and Boost::ASIO. [viitattu 25.10.2015]. Saatavissa: https://honghongcs.wordpress.com/2011/12/30/asynchronous-io-and-boostasio-1/.

[23] Microsoft App Developer Agreement. MSDN. [viitattu 25.10.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/windows/apps/hh694058.aspx.

[24] Microsoft Analytics. MSDN. [viitattu 25.10.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/windows/apps/mt148522.aspx.

[25] Microsoft The app certification process. MSDN. [viitattu 25.10.2015]. Saatavissa: https://msdn.microsoft.com/library/windows/apps/mt148554.aspx.

[26] Microsoft App capability declarations. MSDN. [viitattu 25.10.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/windows/apps/hh464936.aspx.

[27] Microsoft Sideload Windows Store Apps in Windows 8. [viitattu 25.10.2015]. Saatavissa: https://technet.microsoft.com/en-us/library/dn613831.aspx.

[28] Hunter, B. Windows 8.1 Update: Sideloading Enhancements. Microsoft. [viitattu 25.10.2015]. Saatavissa: https://blogs.windows.com/itpro/2014/04/03/windows-8-1-update-sideloading-enhancements/.

[29] Windows 8.1 - 5000 New APIs! [viitattu 5.11.2015]. Saatavissa: http://www.i-programmer.info/news/177-windows-8/6037-windows-81-5000-new-apis.html.

[30] WinRTXamlToolkit. [viitattu 25.10.2015]. Saatavissa: https://github.com/xyzzer/WinRTXamlToolkit.

[31] Microsoft Bing Maps. MSDN. [viitattu 25.10.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/dd877180.aspx.

[32] SqLite Documentation. [viitattu 25.10.2015]. Saatavissa: https://www.sqlite.org/docs.html.

[33] Pakarinen, J. Resurssien synkronointi liikkuvan myyjän tablet-laitteelle : diplomityö. Tampere 2014, Tampereen teknillinen yliopisto. 60 lehteä

[34] Sprott, D. & Wilkes, L. Understanding Service-Oriented Architecture. MSDN. [viitattu 6.11.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/aa480021.aspx.

[35] Microsoft What Is Windows Communication Foundation? MSDN. [viitattu 6.11.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/vstudio/ms731082(v=vs.90).aspx.

[36] What is Entity Framework? [viitattu 6.11.2015]. Saatavissa: http://www.entityframeworktutorial.net/what-is-entityframework.aspx.

[37] Ghoda, A. Windows 8 MVVM Patterns Revealed : Covers both C# and JavaScript. 2012, XXII, 172 p

[38] MVVM Compared To MVC and MVP. [viitattu 25.10.2015]. Saatavissa: http://geekswithblogs.net/dlussier/archive/2009/11/21/136454.aspx.

[39] Implementing the MVVM Pattern. [viitattu 25.10.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/gg405484.

[40] Microsoft Visual Studio 2010 SQL Server Database Projects. MSDN. [viitattu 25.10.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/ff678491.

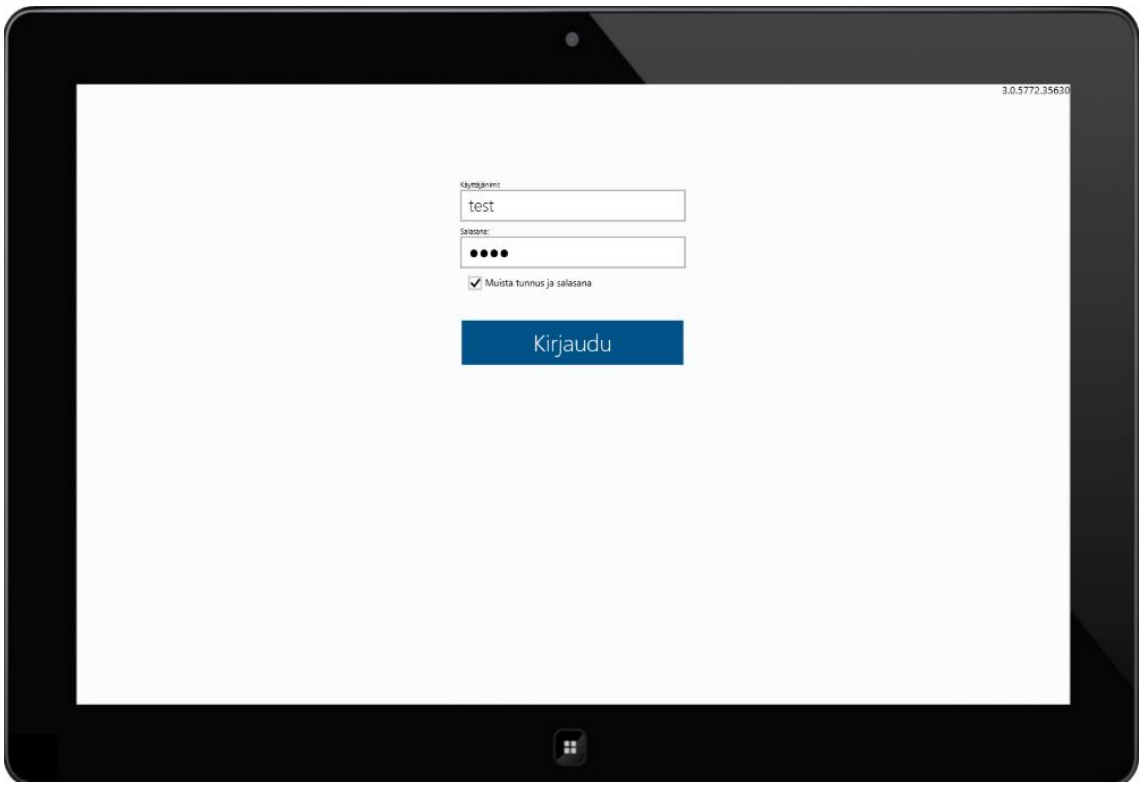
[41] Microsoft SemanticZoom class. MSDN. [viitattu 25.10.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.xaml.controls.semanticzoom.aspx.

[42] Microsoft Globalizing your app. MSDN. [viitattu 25.10.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/windows/apps/xaml/hh965328.aspx.

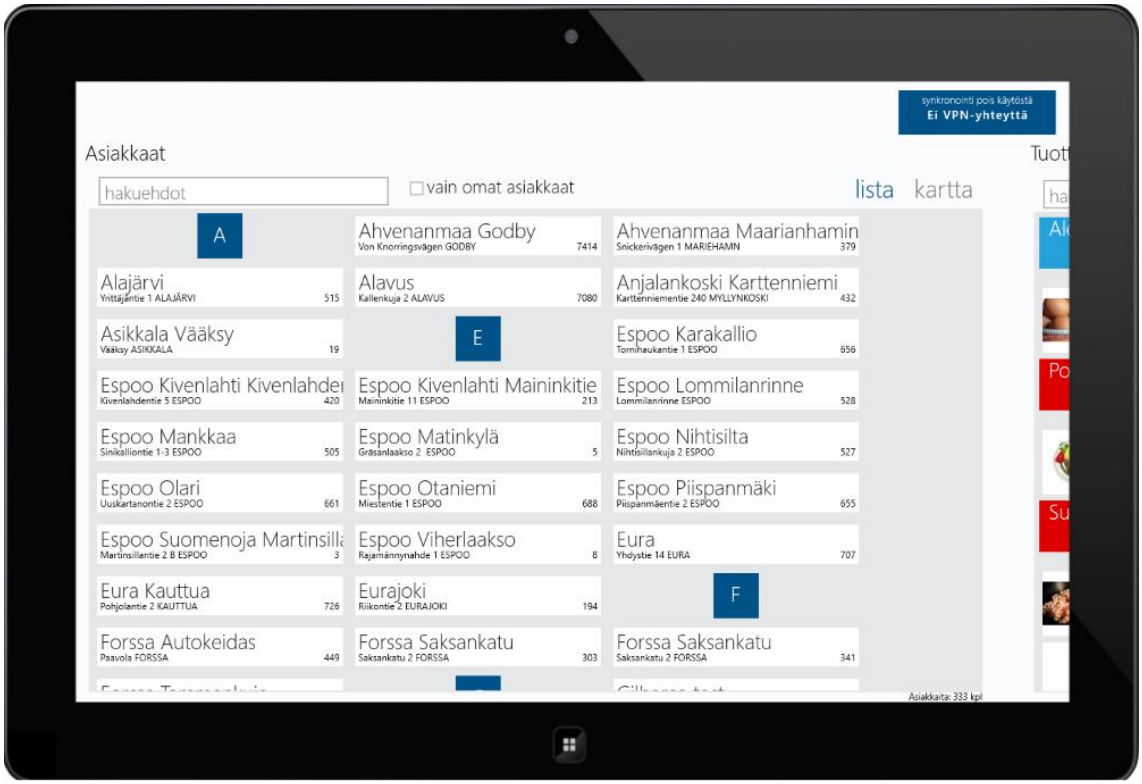
[43] Eatech Portable referenssi - Prima Pet Premium Oy. Eatech. [viitattu 8.11.2015]. Saatavissa: http://eatech.fi/fi/ppp-myyntisovellus-asiakaskertomus.

[44] Eatech Portable referenssi - Miraculos Oy. [viitattu 8.11.2015]. Saatavissa: http://eatech.fi/fi/miraculos-asiakaskertomus.

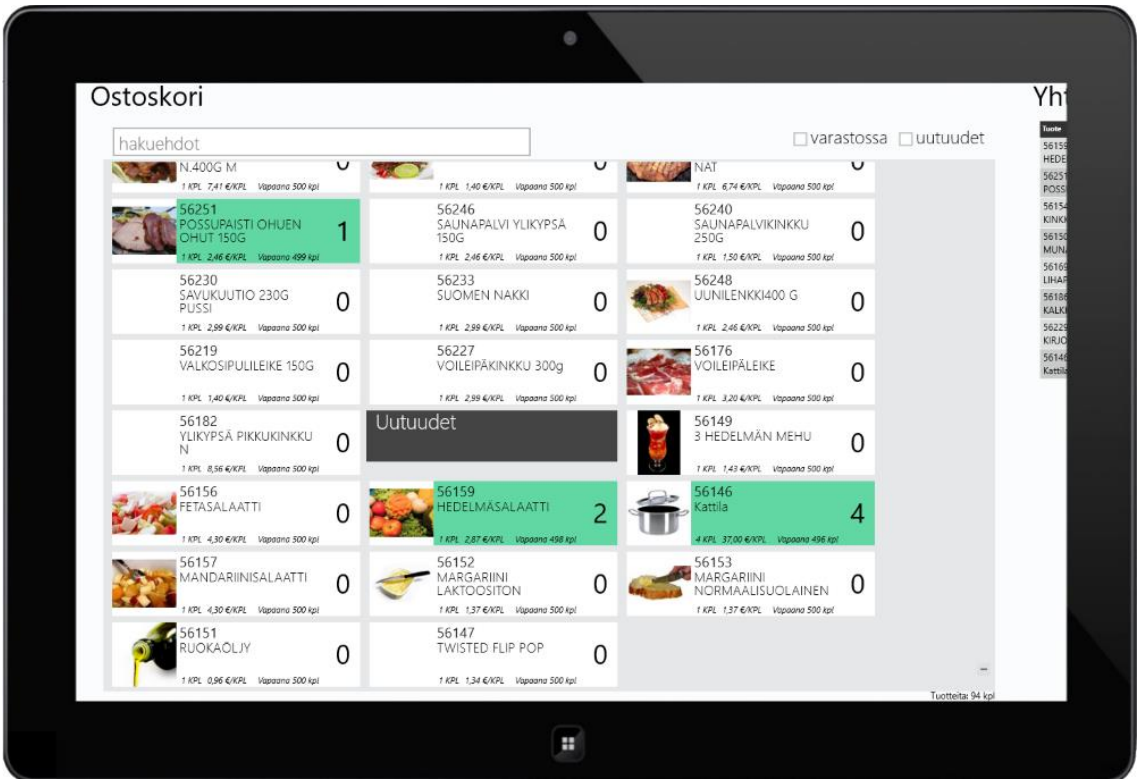
LIITE A: EATECH PORTABLE – KÄYTTÖLIITTYMÄKUVAT



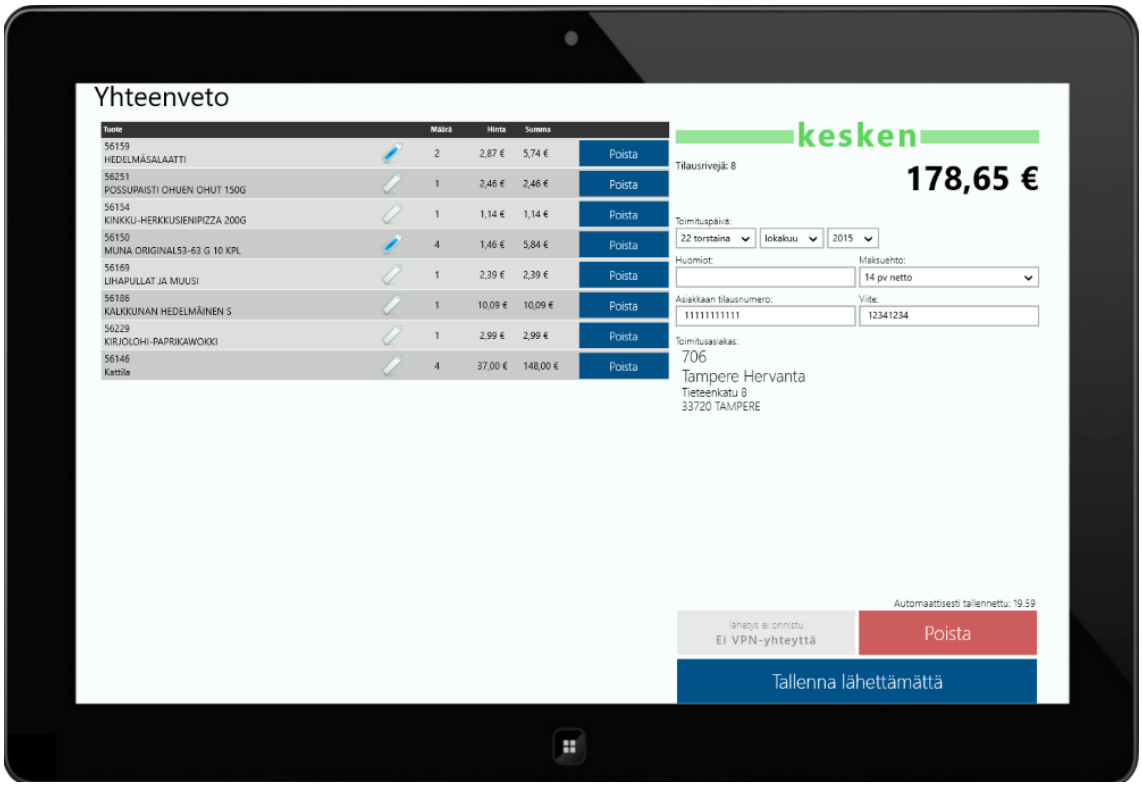
Kuva 1: Kirjautumisnäkymä



Kuva 2: Asiakaslistaus



Kuva 3: Tuotelistaus



Kuva 4: Yhteenveto